(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
12 July 2001 (12.07.2001)

PCT

(10) International Publication Number
WO 01/50598 A2

(51) International Patent Classification⁷: H03G 5/00

(21) International Application Number: PCT/US01/00224

(22) International Filing Date: 4 January 2001 (04.01.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/174,397 5 January 2000 (05.01.2000) US

(71) Applicant *(for all designated States except US)*: **TRUE DIMENSIONAL SOUND, INC.** [US/US]; Suite 404, 1450 Madruga Avenue, Coral Gables, FL 33146 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **SCARPINO, Frank, A.** [US/US]; 1984 Indian Head Road, Dayton, OH 45459 (US). **GARCIA, Arturo, H.** [US/US]; 4443 S.W. 11th Street, Miami, FL 33134 (US). **BLANDFORD, Scott, M.** [US/US]; 154 N. Ft. Thomas Avenue, Ft. Thomas, KY 41075 (US).

(74) Agents: **SHOWALTER, Robert, L.** et al.; King and Schickli PLLC, 247 North Broadway, Lexington, KY 40507 (US).

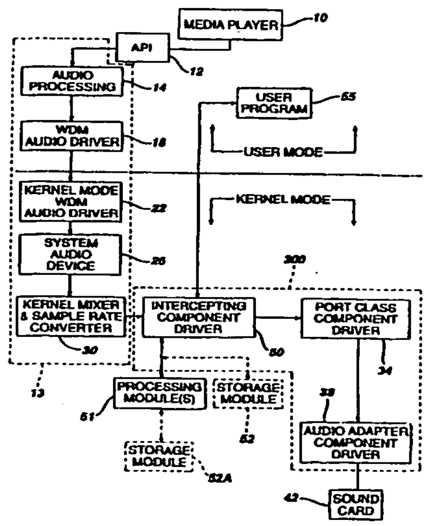
(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *Without international search report and to be republished upon receipt of that report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: AUDIO DEVICE DRIVER



(57) Abstract: A device driver for a computer operating system. The operating system is adapted to receive digitized audio data from a media source. The device driver comprises an intercepting component driver which is capable of initiating at least one of sound modification processing of the digitized data and storage of the digitized data.

WO 01/50598 A2

AUDIO DEVICE DRIVER

Authorization under 37 CFR Sec. 1.71(e)

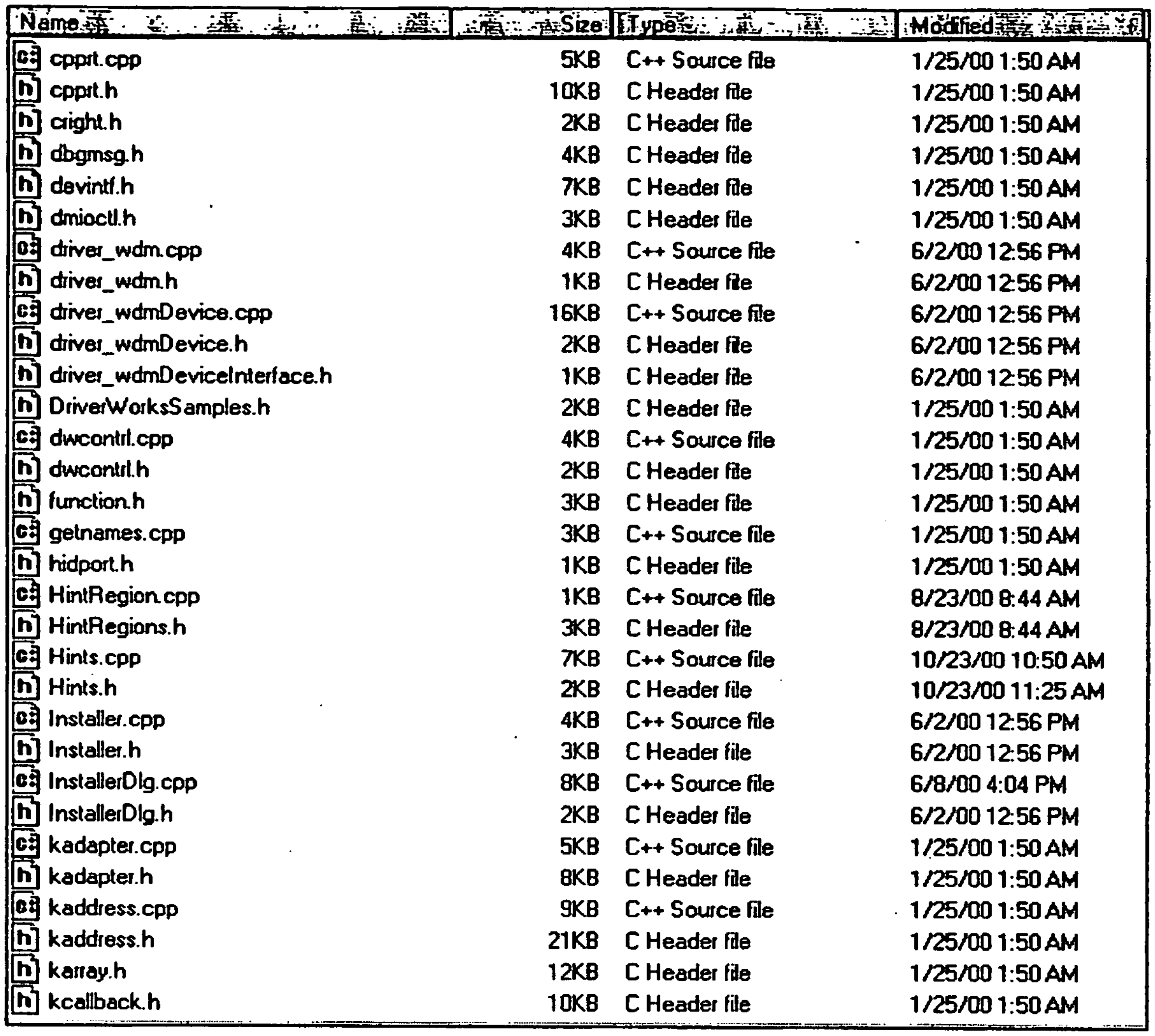
Cross-Reference to Related Application

This application claims the priority of Provisional Application U.S. Serial No. 60/174,397, filed January 5, 2000, the disclosure of which is incorporated herein by reference.

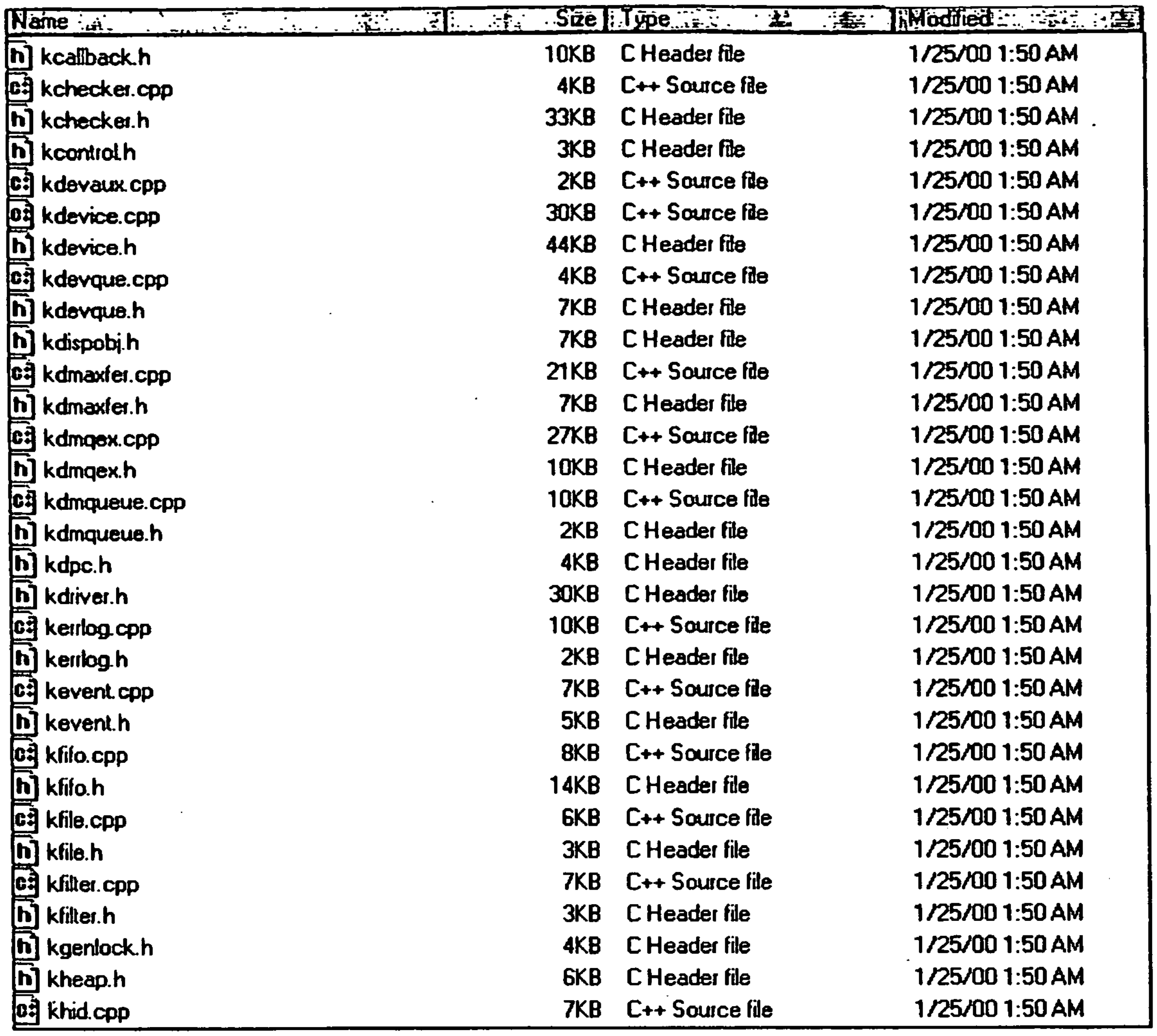
Reference to Computer Program Listing Appendix

Being submitted concurrently with the filing of this application are two compact disks, each containing the following computer program listings; all listings set out below and contained on each disk are incorporated herein by reference:

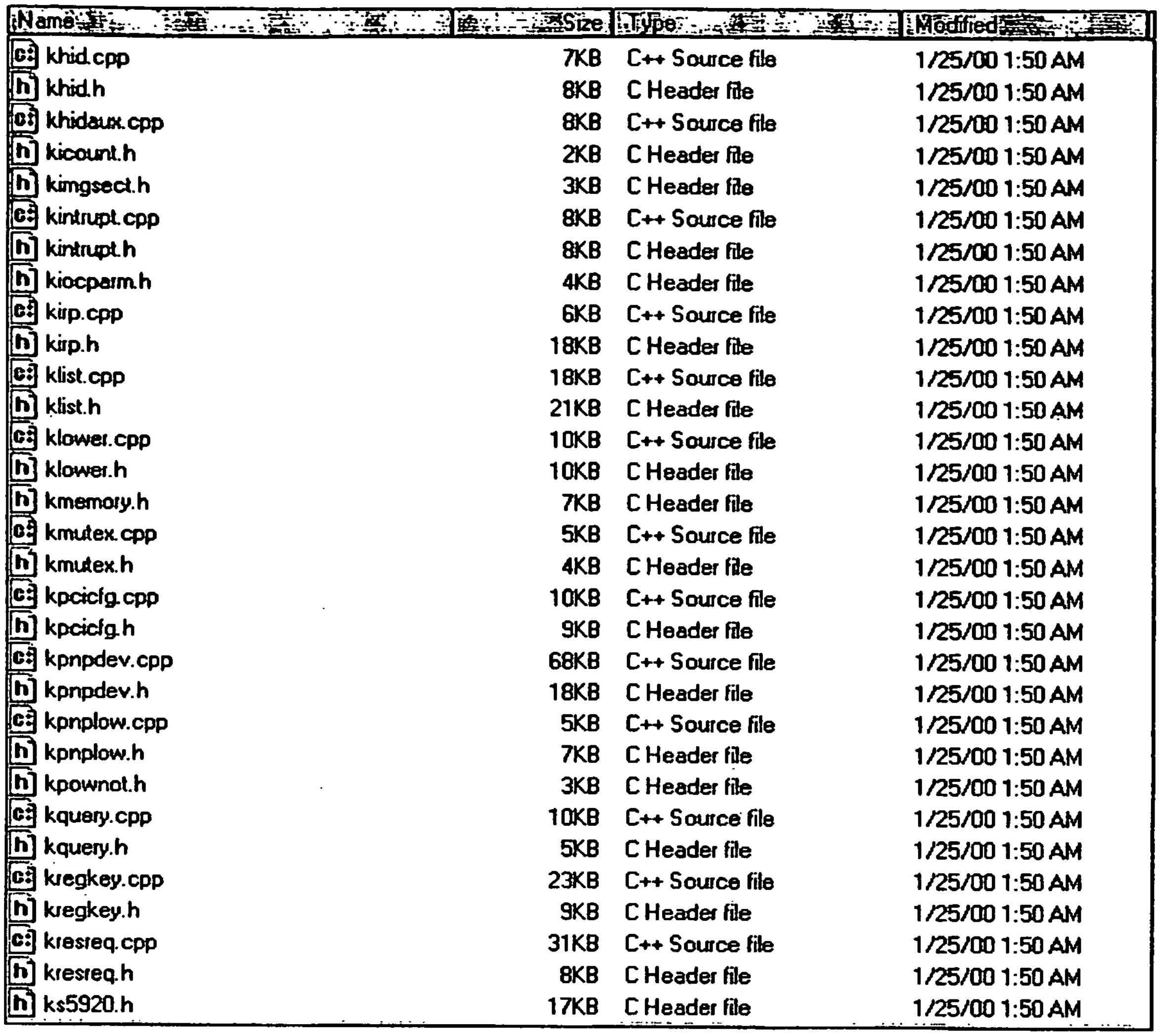
TDS Kernel Mode Code Listings Windows 2000



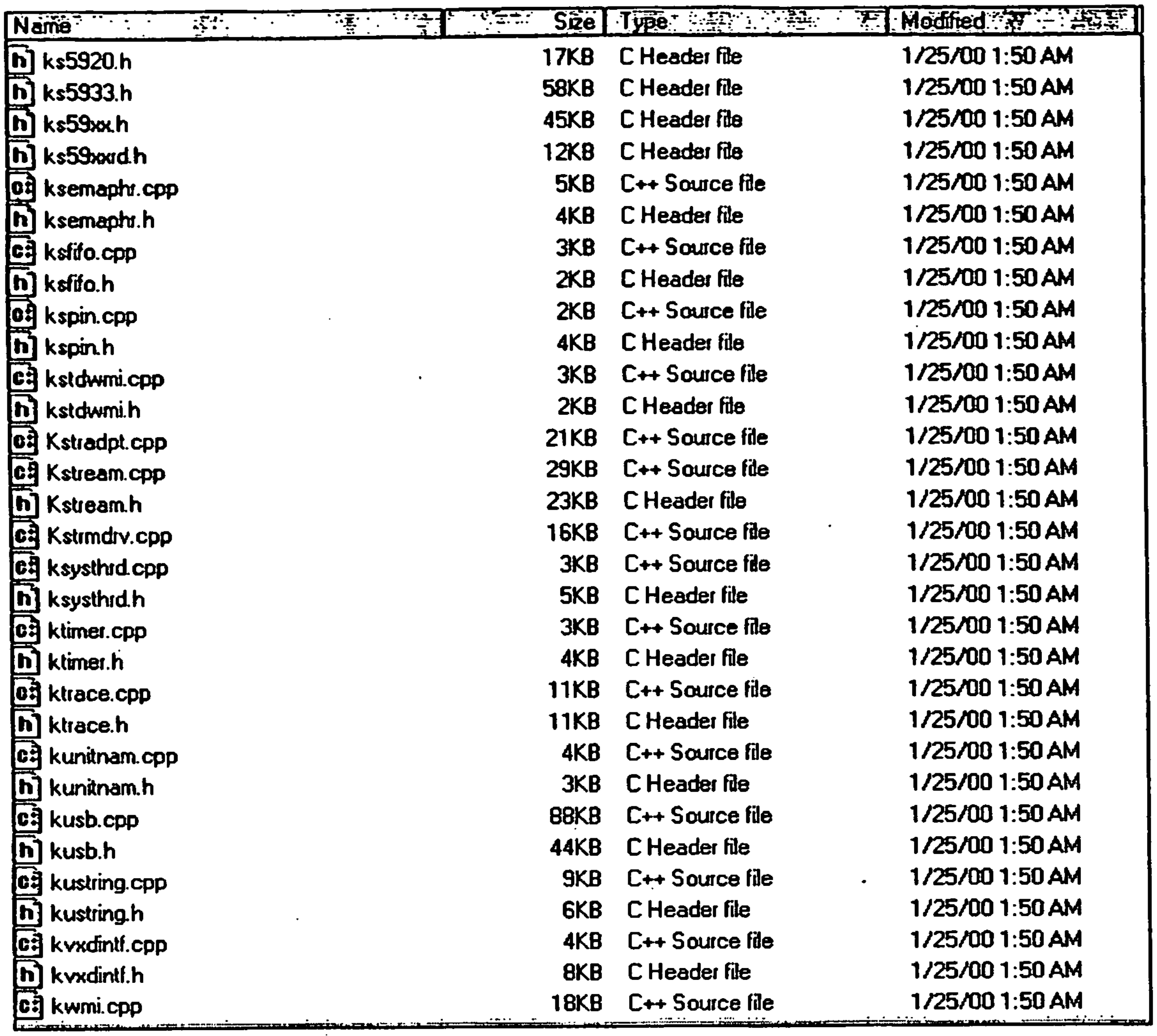
Name	Size	Type	Modified
cpprt.cpp	5KB	C++ Source file	1/25/00 1:50 AM
cpprt.h	10KB	C Header file	1/25/00 1:50 AM
cright.h	2KB	C Header file	1/25/00 1:50 AM
dbgmsg.h	4KB	C Header file	1/25/00 1:50 AM
devintf.h	7KB	C Header file	1/25/00 1:50 AM
dmioctl.h	3KB	C Header file	1/25/00 1:50 AM
driver_wdm.cpp	4KB	C++ Source file	6/2/00 12:56 PM
driver_wdm.h	1KB	C Header file	6/2/00 12:56 PM
driver_wdmDevice.cpp	16KB	C++ Source file	6/2/00 12:56 PM
driver_wdmDevice.h	2KB	C Header file	6/2/00 12:56 PM
driver_wdmDeviceInterface.h	1KB	C Header file	6/2/00 12:56 PM
DriverWorksSamples.h	2KB	C Header file	1/25/00 1:50 AM
dwcontrl.cpp	4KB	C++ Source file	1/25/00 1:50 AM
dwcontrl.h	2KB	C Header file	1/25/00 1:50 AM
function.h	3KB	C Header file	1/25/00 1:50 AM
getnames.cpp	3KB	C++ Source file	1/25/00 1:50 AM
hidport.h	1KB	C Header file	1/25/00 1:50 AM
HintRegion.cpp	1KB	C++ Source file	8/23/00 8:44 AM
HintRegions.h	3KB	C Header file	8/23/00 8:44 AM
Hints.cpp	7KB	C++ Source file	10/23/00 10:50 AM
Hints.h	2KB	C Header file	10/23/00 11:25 AM
Installer.cpp	4KB	C++ Source file	6/2/00 12:56 PM
Installer.h	3KB	C Header file	6/2/00 12:56 PM
InstallerDlg.cpp	8KB	C++ Source file	6/8/00 4:04 PM
InstallerDlg.h	2KB	C Header file	6/2/00 12:56 PM
kadapter.cpp	5KB	C++ Source file	1/25/00 1:50 AM
kadapter.h	8KB	C Header file	1/25/00 1:50 AM
kaddress.cpp	9KB	C++ Source file	1/25/00 1:50 AM
kaddress.h	21KB	C Header file	1/25/00 1:50 AM
karray.h	12KB	C Header file	1/25/00 1:50 AM
kcallback.h	10KB	C Header file	1/25/00 1:50 AM



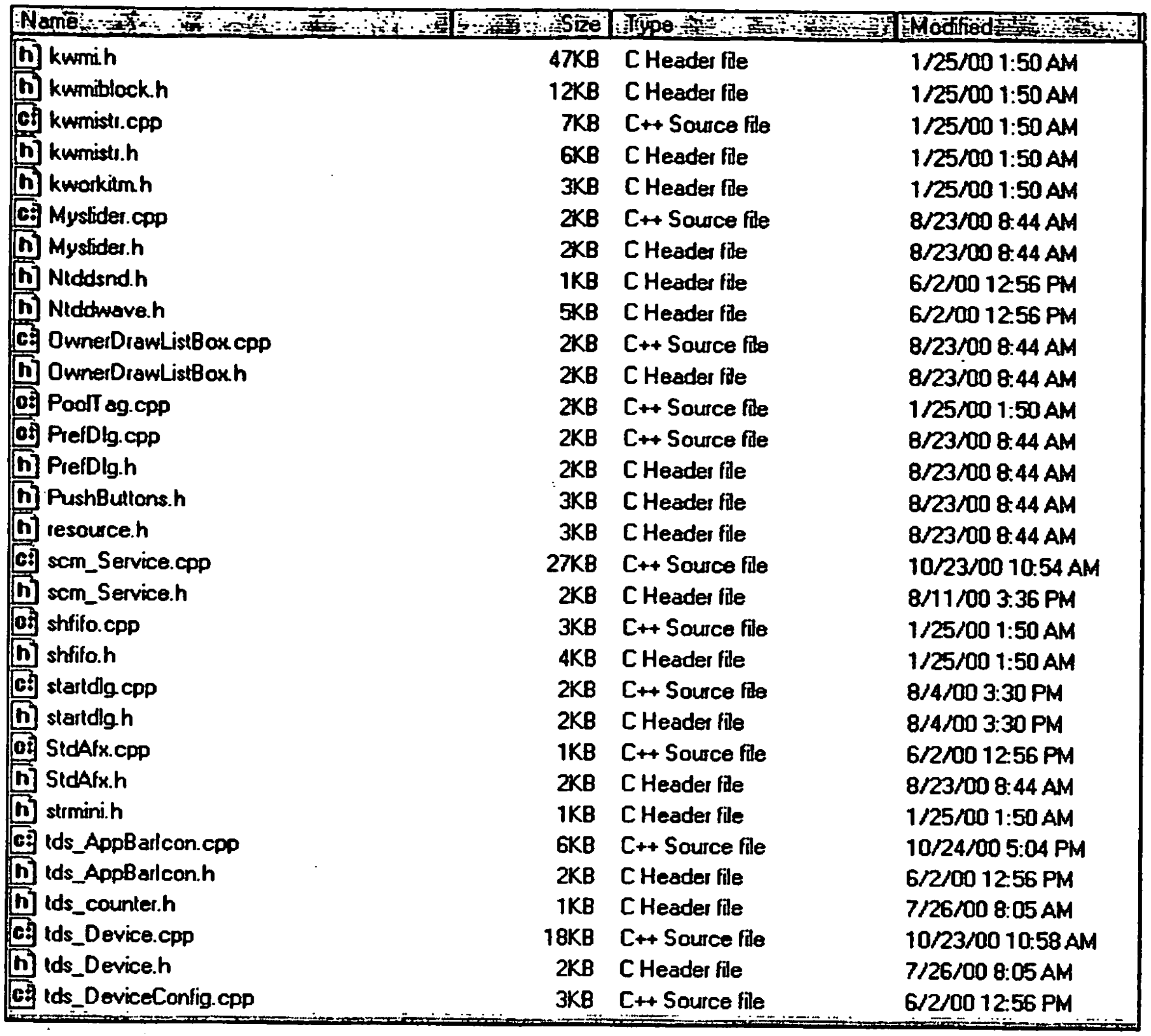
Name	Size	Type	Modified
kcallback.h	10KB	C Header file	1/25/00 1:50 AM
kchecker.cpp	4KB	C++ Source file	1/25/00 1:50 AM
kchecker.h	33KB	C Header file	1/25/00 1:50 AM
kcontrol.h	3KB	C Header file	1/25/00 1:50 AM
kdevaux.cpp	2KB	C++ Source file	1/25/00 1:50 AM
kdevice.cpp	30KB	C++ Source file	1/25/00 1:50 AM
kdevice.h	44KB	C Header file	1/25/00 1:50 AM
kdevque.cpp	4KB	C++ Source file	1/25/00 1:50 AM
kdevque.h	7KB	C Header file	1/25/00 1:50 AM
kdispobj.h	7KB	C Header file	1/25/00 1:50 AM
kdmaxfer.cpp	21KB	C++ Source file	1/25/00 1:50 AM
kdmaxfer.h	7KB	C Header file	1/25/00 1:50 AM
kdmqex.cpp	27KB	C++ Source file	1/25/00 1:50 AM
kdmqex.h	10KB	C Header file	1/25/00 1:50 AM
kdmqueue.cpp	10KB	C++ Source file	1/25/00 1:50 AM
kdmqueue.h	2KB	C Header file	1/25/00 1:50 AM
kdpc.h	4KB	C Header file	1/25/00 1:50 AM
kdriver.h	30KB	C Header file	1/25/00 1:50 AM
kerrlog.cpp	10KB	C++ Source file	1/25/00 1:50 AM
kerrlog.h	2KB	C Header file	1/25/00 1:50 AM
kevent.cpp	7KB	C++ Source file	1/25/00 1:50 AM
kevent.h	5KB	C Header file	1/25/00 1:50 AM
kfifo.cpp	8KB	C++ Source file	1/25/00 1:50 AM
kfifo.h	14KB	C Header file	1/25/00 1:50 AM
kfile.cpp	6KB	C++ Source file	1/25/00 1:50 AM
kfile.h	3KB	C Header file	1/25/00 1:50 AM
kfilter.cpp	7KB	C++ Source file	1/25/00 1:50 AM
kfilter.h	3KB	C Header file	1/25/00 1:50 AM
kgenlock.h	4KB	C Header file	1/25/00 1:50 AM
kheap.h	6KB	C Header file	1/25/00 1:50 AM
khid.cpp	7KB	C++ Source file	1/25/00 1:50 AM



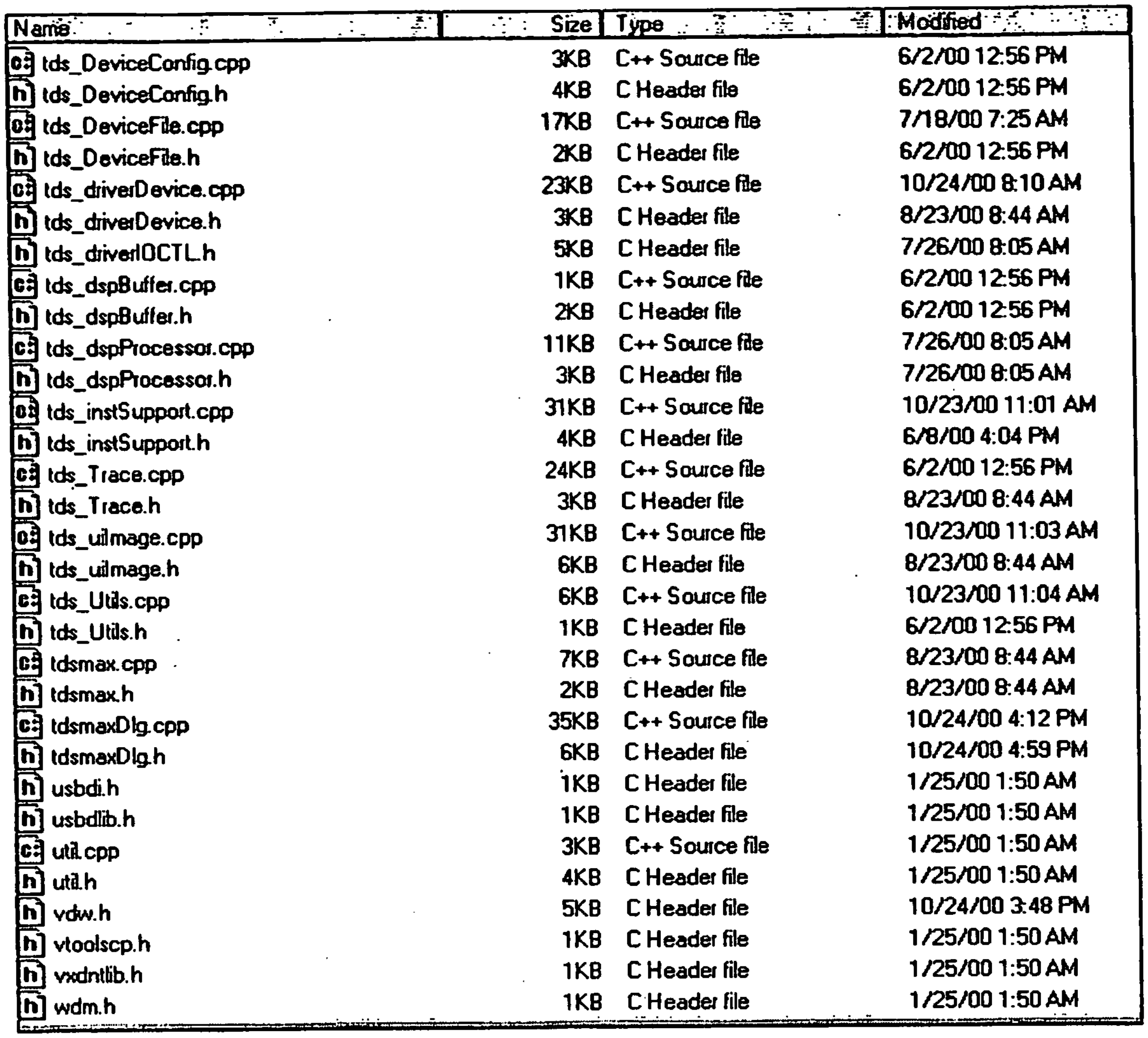
Name	Size	Type	Modified
khid.cpp	7KB	C++ Source file	1/25/00 1:50 AM
khid.h	8KB	C Header file	1/25/00 1:50 AM
khidaux.cpp	8KB	C++ Source file	1/25/00 1:50 AM
kicount.h	2KB	C Header file	1/25/00 1:50 AM
kimgsect.h	3KB	C Header file	1/25/00 1:50 AM
kintrupt.cpp	8KB	C++ Source file	1/25/00 1:50 AM
kintrupt.h	8KB	C Header file	1/25/00 1:50 AM
kiocparm.h	4KB	C Header file	1/25/00 1:50 AM
kirp.cpp	6KB	C++ Source file	1/25/00 1:50 AM
kirp.h	18KB	C Header file	1/25/00 1:50 AM
klist.cpp	18KB	C++ Source file	1/25/00 1:50 AM
klist.h	21KB	C Header file	1/25/00 1:50 AM
klower.cpp	10KB	C++ Source file	1/25/00 1:50 AM
klower.h	10KB	C Header file	1/25/00 1:50 AM
kmemory.h	7KB	C Header file	1/25/00 1:50 AM
kmutex.cpp	5KB	C++ Source file	1/25/00 1:50 AM
kmutex.h	4KB	C Header file	1/25/00 1:50 AM
kpcicfg.cpp	10KB	C++ Source file	1/25/00 1:50 AM
kpcicfg.h	9KB	C Header file	1/25/00 1:50 AM
kpnpdev.cpp	68KB	C++ Source file	1/25/00 1:50 AM
kpnpdev.h	18KB	C Header file	1/25/00 1:50 AM
kpnplow.cpp	5KB	C++ Source file	1/25/00 1:50 AM
kpnplow.h	7KB	C Header file	1/25/00 1:50 AM
kpownot.h	3KB	C Header file	1/25/00 1:50 AM
kquery.cpp	10KB	C++ Source file	1/25/00 1:50 AM
kquery.h	5KB	C Header file	1/25/00 1:50 AM
kregkey.cpp	23KB	C++ Source file	1/25/00 1:50 AM
kregkey.h	9KB	C Header file	1/25/00 1:50 AM
kresreq.cpp	31KB	C++ Source file	1/25/00 1:50 AM
kresreq.h	8KB	C Header file	1/25/00 1:50 AM
ks5920.h	17KB	C Header file	1/25/00 1:50 AM



Name	Size	Type	Modified
ks5920.h	17KB	C Header file	1/25/00 1:50 AM
ks5933.h	58KB	C Header file	1/25/00 1:50 AM
ks59xx.h	45KB	C Header file	1/25/00 1:50 AM
ks59xxrd.h	12KB	C Header file	1/25/00 1:50 AM
ksemaphr.cpp	5KB	C++ Source file	1/25/00 1:50 AM
ksemaphr.h	4KB	C Header file	1/25/00 1:50 AM
ksfifo.cpp	3KB	C++ Source file	1/25/00 1:50 AM
ksfifo.h	2KB	C Header file	1/25/00 1:50 AM
kspin.cpp	2KB	C++ Source file	1/25/00 1:50 AM
kspin.h	4KB	C Header file	1/25/00 1:50 AM
kstdwmi.cpp	3KB	C++ Source file	1/25/00 1:50 AM
kstdwmi.h	2KB	C Header file	1/25/00 1:50 AM
Kstradpt.cpp	21KB	C++ Source file	1/25/00 1:50 AM
Kstream.cpp	29KB	C++ Source file	1/25/00 1:50 AM
Kstream.h	23KB	C Header file	1/25/00 1:50 AM
Kstrmdrv.cpp	16KB	C++ Source file	1/25/00 1:50 AM
ksysthrd.cpp	3KB	C++ Source file	1/25/00 1:50 AM
ksysthrd.h	5KB	C Header file	1/25/00 1:50 AM
ktimer.cpp	3KB	C++ Source file	1/25/00 1:50 AM
ktimer.h	4KB	C Header file	1/25/00 1:50 AM
ktrace.cpp	11KB	C++ Source file	1/25/00 1:50 AM
ktrace.h	11KB	C Header file	1/25/00 1:50 AM
kunitnam.cpp	4KB	C++ Source file	1/25/00 1:50 AM
kunitnam.h	3KB	C Header file	1/25/00 1:50 AM
kusb.cpp	88KB	C++ Source file	1/25/00 1:50 AM
kusb.h	44KB	C Header file	1/25/00 1:50 AM
kustring.cpp	9KB	C++ Source file	1/25/00 1:50 AM
kustring.h	6KB	C Header file	1/25/00 1:50 AM
kvxdintf.cpp	4KB	C++ Source file	1/25/00 1:50 AM
kvxdintf.h	8KB	C Header file	1/25/00 1:50 AM
kwmi.cpp	18KB	C++ Source file	1/25/00 1:50 AM



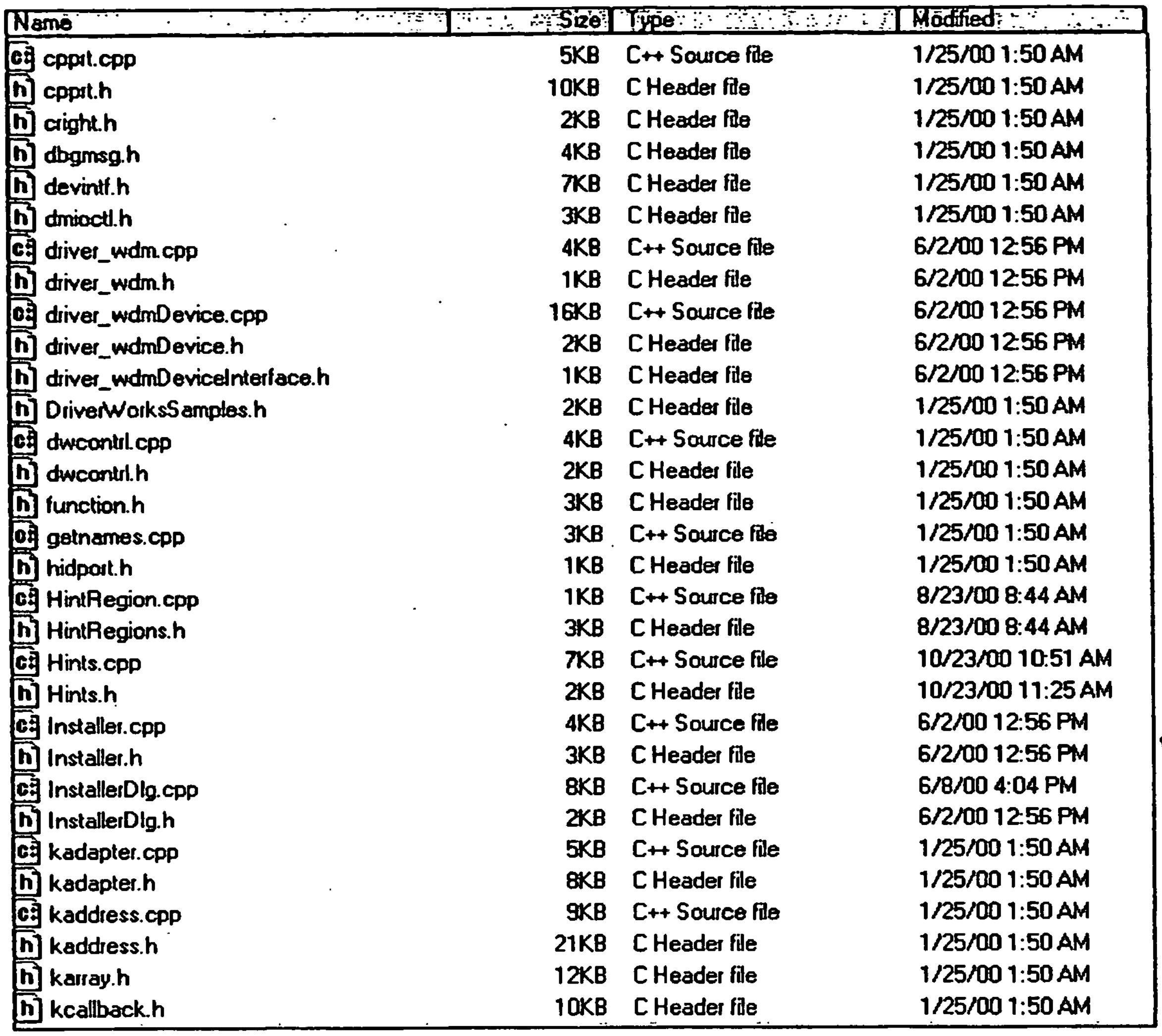
Name	Size	Type	Modified
kwmi.h	47KB	C Header file	1/25/00 1:50 AM
kwmiblock.h	12KB	C Header file	1/25/00 1:50 AM
kwmistr.cpp	7KB	C++ Source file	1/25/00 1:50 AM
kwmistr.h	6KB	C Header file	1/25/00 1:50 AM
kworkitm.h	3KB	C Header file	1/25/00 1:50 AM
Myslider.cpp	2KB	C++ Source file	8/23/00 8:44 AM
Myslider.h	2KB	C Header file	8/23/00 8:44 AM
Ntddsnd.h	1KB	C Header file	6/2/00 12:56 PM
Ntddwave.h	5KB	C Header file	6/2/00 12:56 PM
OwnerDrawListBox.cpp	2KB	C++ Source file	8/23/00 8:44 AM
OwnerDrawListBox.h	2KB	C Header file	8/23/00 8:44 AM
PoolTag.cpp	2KB	C++ Source file	1/25/00 1:50 AM
PrefDlg.cpp	2KB	C++ Source file	8/23/00 8:44 AM
PrefDlg.h	2KB	C Header file	8/23/00 8:44 AM
PushButtons.h	3KB	C Header file	8/23/00 8:44 AM
resource.h	3KB	C Header file	8/23/00 8:44 AM
scm_Service.cpp	27KB	C++ Source file	10/23/00 10:54 AM
scm_Service.h	2KB	C Header file	8/11/00 3:36 PM
shfifo.cpp	3KB	C++ Source file	1/25/00 1:50 AM
shfifo.h	4KB	C Header file	1/25/00 1:50 AM
startdlg.cpp	2KB	C++ Source file	8/4/00 3:30 PM
startdlg.h	2KB	C Header file	8/4/00 3:30 PM
StdAfx.cpp	1KB	C++ Source file	6/2/00 12:56 PM
StdAfx.h	2KB	C Header file	8/23/00 8:44 AM
strmini.h	1KB	C Header file	1/25/00 1:50 AM
tds_AppBarIcon.cpp	6KB	C++ Source file	10/24/00 5:04 PM
tds_AppBarIcon.h	2KB	C Header file	6/2/00 12:56 PM
tds_counter.h	1KB	C Header file	7/26/00 8:05 AM
tds_Device.cpp	18KB	C++ Source file	10/23/00 10:58 AM
tds_Device.h	2KB	C Header file	7/26/00 8:05 AM
tds_DeviceConfig.cpp	3KB	C++ Source file	6/2/00 12:56 PM



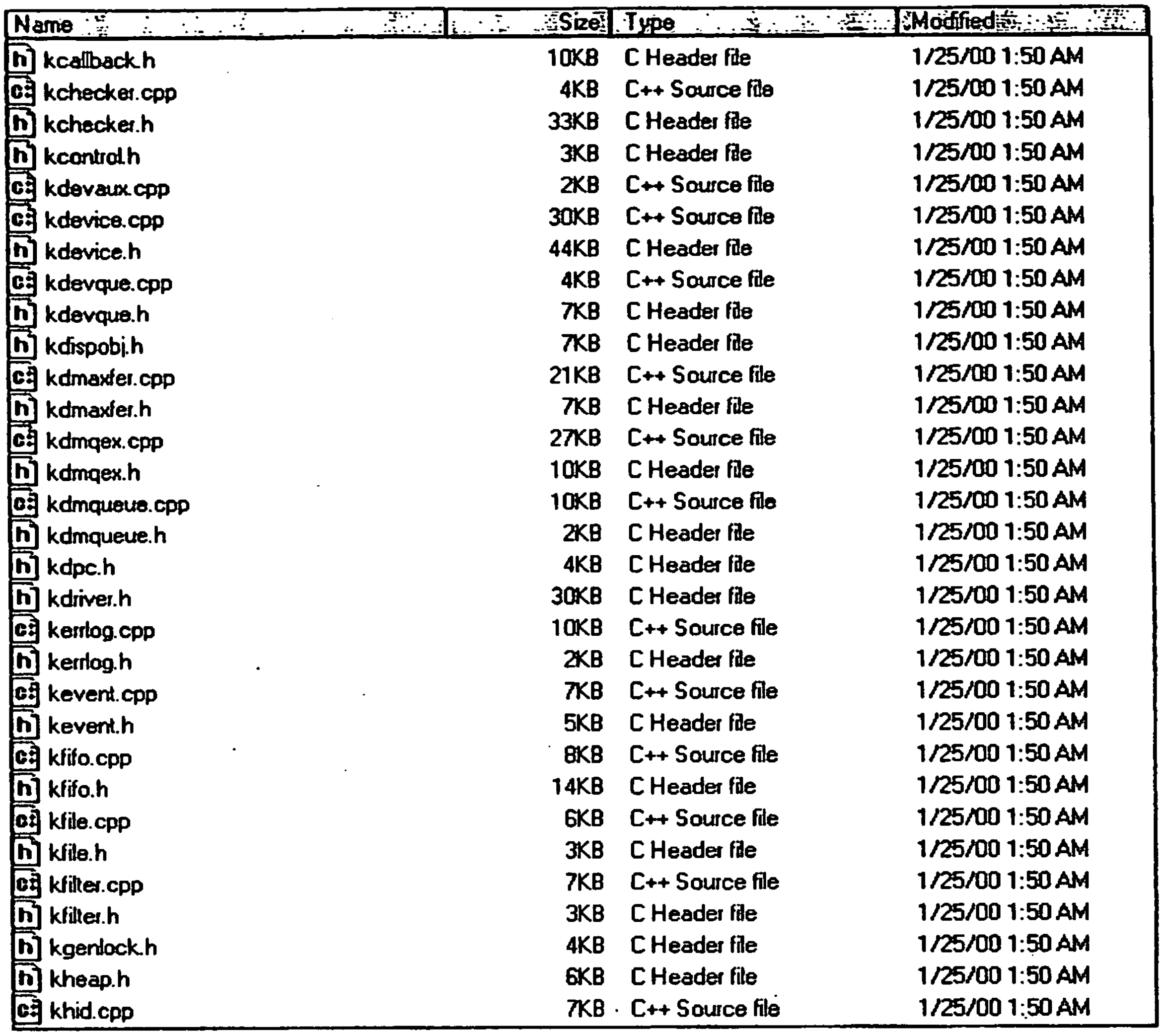
Name	Size	Type	Modified
tds_DeviceConfig.cpp	3KB	C++ Source file	6/2/00 12:56 PM
tds_DeviceConfig.h	4KB	C Header file	6/2/00 12:56 PM
tds_DeviceFile.cpp	17KB	C++ Source file	7/18/00 7:25 AM
tds_DeviceFile.h	2KB	C Header file	6/2/00 12:56 PM
tds_driverDevice.cpp	23KB	C++ Source file	10/24/00 8:10 AM
tds_driverDevice.h	3KB	C Header file	8/23/00 8:44 AM
tds_driverIOCTL.h	5KB	C Header file	7/26/00 8:05 AM
tds_dspBuffer.cpp	1KB	C++ Source file	6/2/00 12:56 PM
tds_dspBuffer.h	2KB	C Header file	6/2/00 12:56 PM
tds_dspProcessor.cpp	11KB	C++ Source file	7/26/00 8:05 AM
tds_dspProcessor.h	3KB	C Header file	7/26/00 8:05 AM
tds_instSupport.cpp	31KB	C++ Source file	10/23/00 11:01 AM
tds_instSupport.h	4KB	C Header file	6/8/00 4:04 PM
tds_Trace.cpp	24KB	C++ Source file	6/2/00 12:56 PM
tds_Trace.h	3KB	C Header file	8/23/00 8:44 AM
tds_uiImage.cpp	31KB	C++ Source file	10/23/00 11:03 AM
tds_uiImage.h	6KB	C Header file	8/23/00 8:44 AM
tds_Utils.cpp	6KB	C++ Source file	10/23/00 11:04 AM
tds_Utils.h	1KB	C Header file	6/2/00 12:56 PM
tdsmax.cpp	7KB	C++ Source file	8/23/00 8:44 AM
tdsmax.h	2KB	C Header file	8/23/00 8:44 AM
tdsmaxDlg.cpp	35KB	C++ Source file	10/24/00 4:12 PM
tdsmaxDlg.h	6KB	C Header file	10/24/00 4:59 PM
usbdi.h	1KB	C Header file	1/25/00 1:50 AM
usbdlib.h	1KB	C Header file	1/25/00 1:50 AM
util.cpp	3KB	C++ Source file	1/25/00 1:50 AM
util.h	4KB	C Header file	1/25/00 1:50 AM
vdw.h	5KB	C Header file	10/24/00 3:48 PM
vtoolscp.h	1KB	C Header file	1/25/00 1:50 AM
vxdntlib.h	1KB	C Header file	1/25/00 1:50 AM
wdm.h	1KB	C Header file	1/25/00 1:50 AM

End Windows 2000

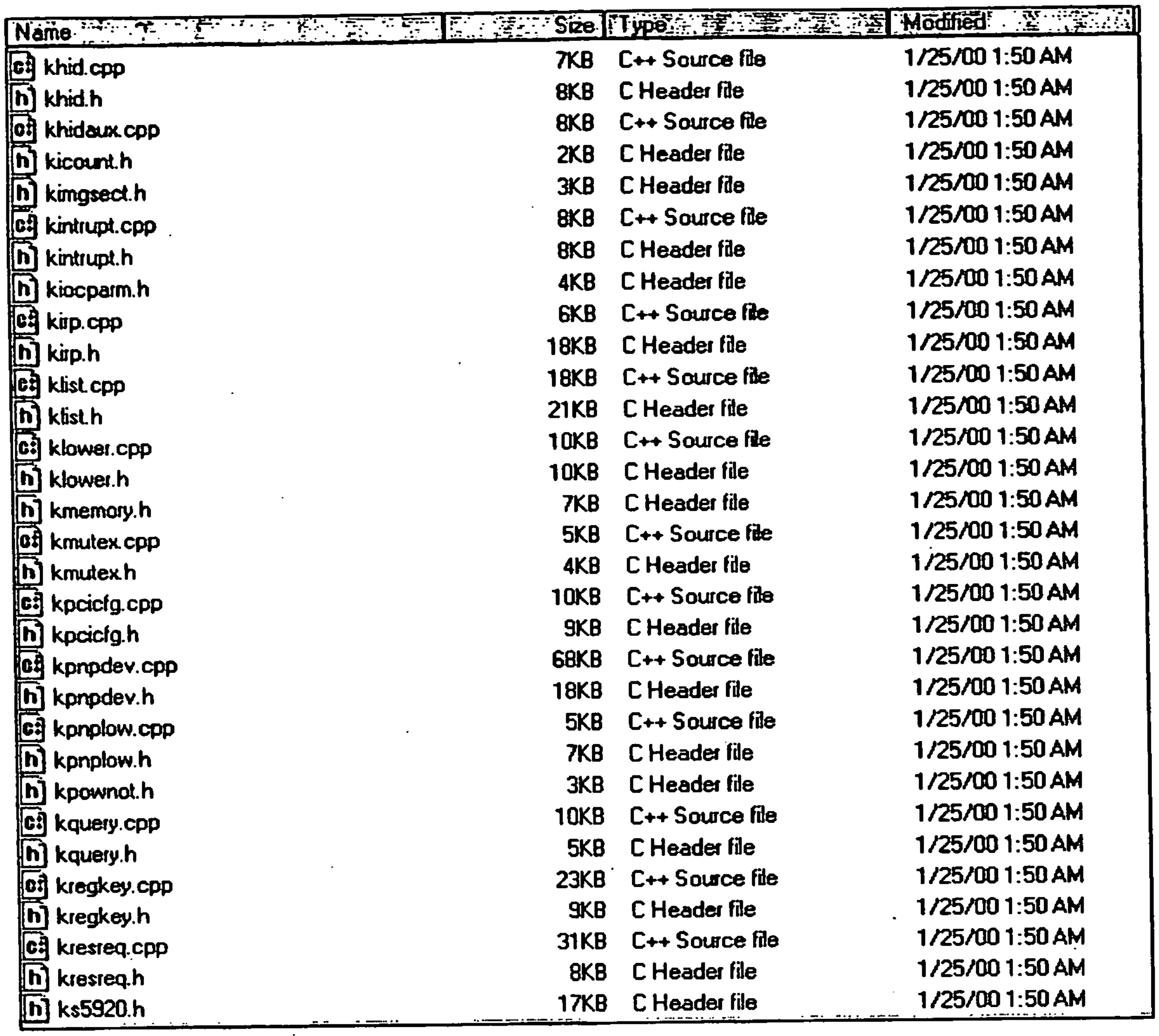
TDS Kernel Mode Source Windows 98



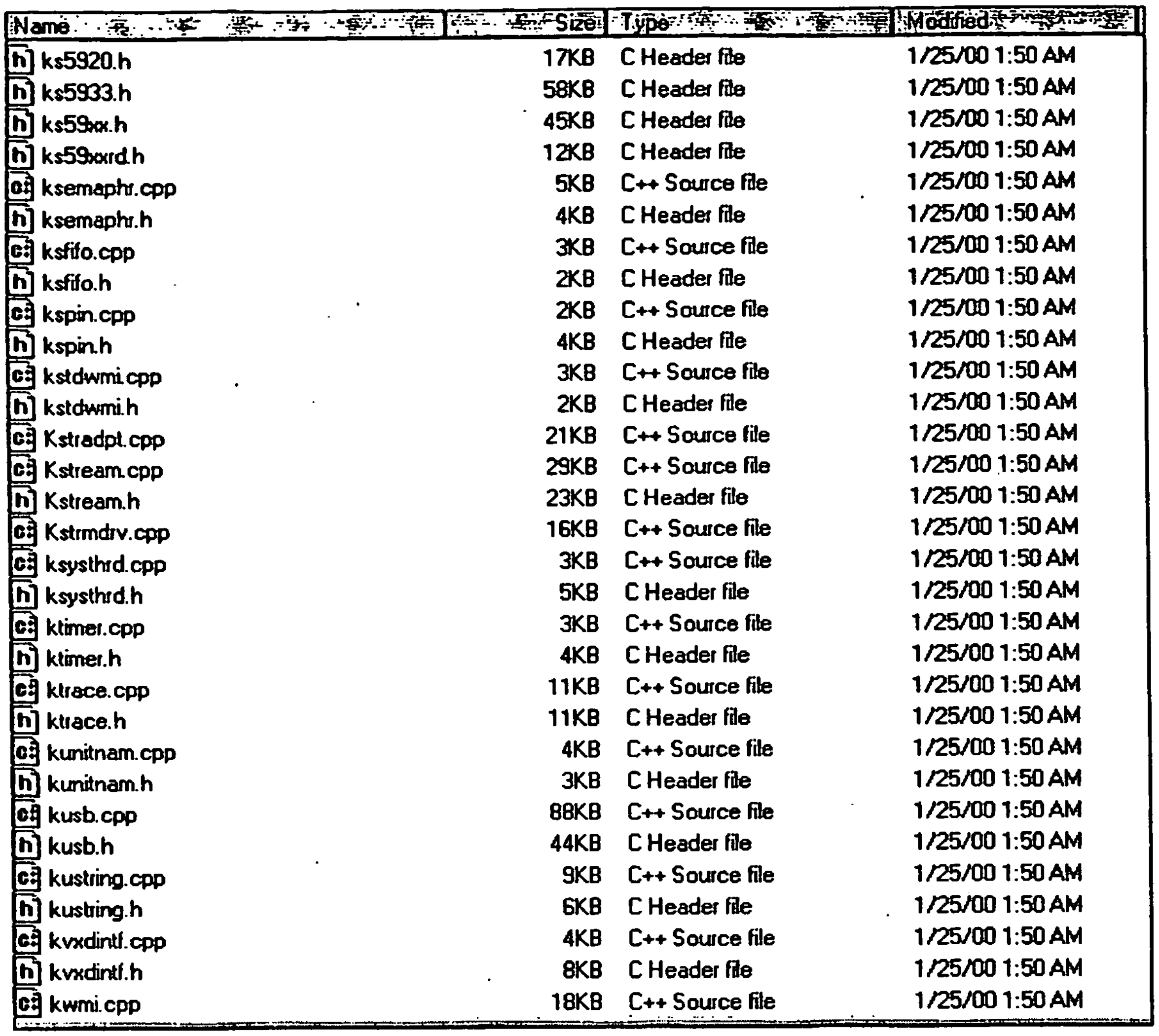
Name	Size	Type	Modified
cpprt.cpp	5KB	C++ Source file	1/25/00 1:50 AM
cpprt.h	10KB	C Header file	1/25/00 1:50 AM
cright.h	2KB	C Header file	1/25/00 1:50 AM
dbgmsg.h	4KB	C Header file	1/25/00 1:50 AM
devintf.h	7KB	C Header file	1/25/00 1:50 AM
dmioctl.h	3KB	C Header file	1/25/00 1:50 AM
driver_wdm.cpp	4KB	C++ Source file	6/2/00 12:56 PM
driver_wdm.h	1KB	C Header file	6/2/00 12:56 PM
driver_wdmDevice.cpp	16KB	C++ Source file	6/2/00 12:56 PM
driver_wdmDevice.h	2KB	C Header file	6/2/00 12:56 PM
driver_wdmDeviceInterface.h	1KB	C Header file	6/2/00 12:56 PM
DriverWorksSamples.h	2KB	C Header file	1/25/00 1:50 AM
dwcontrl.cpp	4KB	C++ Source file	1/25/00 1:50 AM
dwcontrl.h	2KB	C Header file	1/25/00 1:50 AM
function.h	3KB	C Header file	1/25/00 1:50 AM
getnames.cpp	3KB	C++ Source file	1/25/00 1:50 AM
hidport.h	1KB	C Header file	1/25/00 1:50 AM
HintRegion.cpp	1KB	C++ Source file	8/23/00 8:44 AM
HintRegions.h	3KB	C Header file	8/23/00 8:44 AM
Hints.cpp	7KB	C++ Source file	10/23/00 10:51 AM
Hints.h	2KB	C Header file	10/23/00 11:25 AM
Installer.cpp	4KB	C++ Source file	6/2/00 12:56 PM
Installer.h	3KB	C Header file	6/2/00 12:56 PM
InstallerDlg.cpp	8KB	C++ Source file	6/8/00 4:04 PM
InstallerDlg.h	2KB	C Header file	6/2/00 12:56 PM
kadapter.cpp	5KB	C++ Source file	1/25/00 1:50 AM
kadapter.h	8KB	C Header file	1/25/00 1:50 AM
kaddress.cpp	9KB	C++ Source file	1/25/00 1:50 AM
kaddress.h	21KB	C Header file	1/25/00 1:50 AM
karray.h	12KB	C Header file	1/25/00 1:50 AM
kcallback.h	10KB	C Header file	1/25/00 1:50 AM



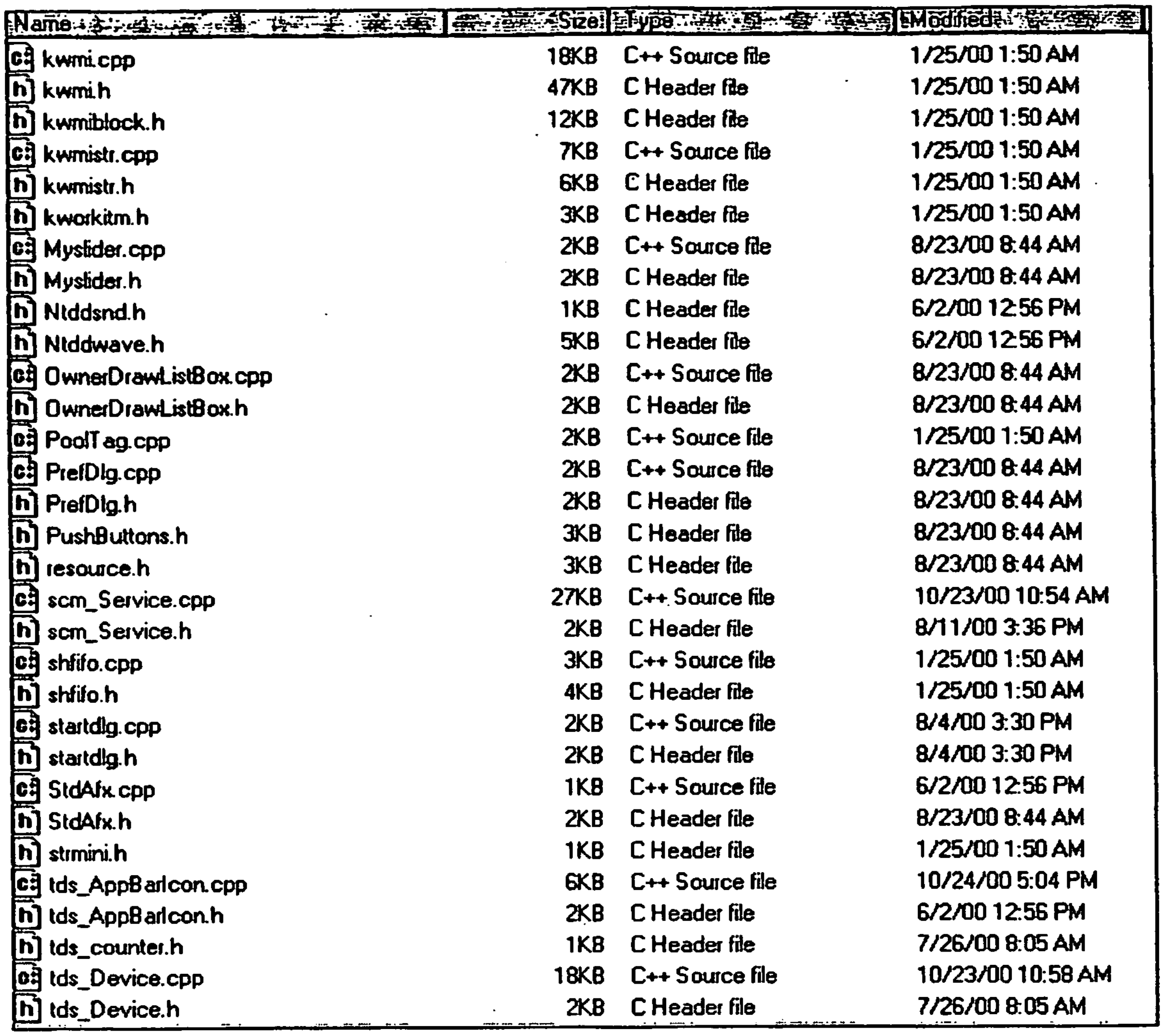
Name	Size	Type	Modified
kcallback.h	10KB	C Header file	1/25/00 1:50 AM
kchecker.cpp	4KB	C++ Source file	1/25/00 1:50 AM
kchecker.h	33KB	C Header file	1/25/00 1:50 AM
kcontrol.h	3KB	C Header file	1/25/00 1:50 AM
kdevaux.cpp	2KB	C++ Source file	1/25/00 1:50 AM
kdevice.cpp	30KB	C++ Source file	1/25/00 1:50 AM
kdevice.h	44KB	C Header file	1/25/00 1:50 AM
kdevque.cpp	4KB	C++ Source file	1/25/00 1:50 AM
kdevque.h	7KB	C Header file	1/25/00 1:50 AM
kdispobj.h	7KB	C Header file	1/25/00 1:50 AM
kdmaxfer.cpp	21KB	C++ Source file	1/25/00 1:50 AM
kdmaxfer.h	7KB	C Header file	1/25/00 1:50 AM
kdmqex.cpp	27KB	C++ Source file	1/25/00 1:50 AM
kdmqex.h	10KB	C Header file	1/25/00 1:50 AM
kdmqueue.cpp	10KB	C++ Source file	1/25/00 1:50 AM
kdmqueue.h	2KB	C Header file	1/25/00 1:50 AM
kdpc.h	4KB	C Header file	1/25/00 1:50 AM
kdriver.h	30KB	C Header file	1/25/00 1:50 AM
kerrlog.cpp	10KB	C++ Source file	1/25/00 1:50 AM
kerrlog.h	2KB	C Header file	1/25/00 1:50 AM
kevent.cpp	7KB	C++ Source file	1/25/00 1:50 AM
kevent.h	5KB	C Header file	1/25/00 1:50 AM
kfifo.cpp	8KB	C++ Source file	1/25/00 1:50 AM
kfifo.h	14KB	C Header file	1/25/00 1:50 AM
kfile.cpp	6KB	C++ Source file	1/25/00 1:50 AM
kfile.h	3KB	C Header file	1/25/00 1:50 AM
kfilter.cpp	7KB	C++ Source file	1/25/00 1:50 AM
kfilter.h	3KB	C Header file	1/25/00 1:50 AM
kgenlock.h	4KB	C Header file	1/25/00 1:50 AM
kheap.h	6KB	C Header file	1/25/00 1:50 AM
khid.cpp	7KB	C++ Source file	1/25/00 1:50 AM



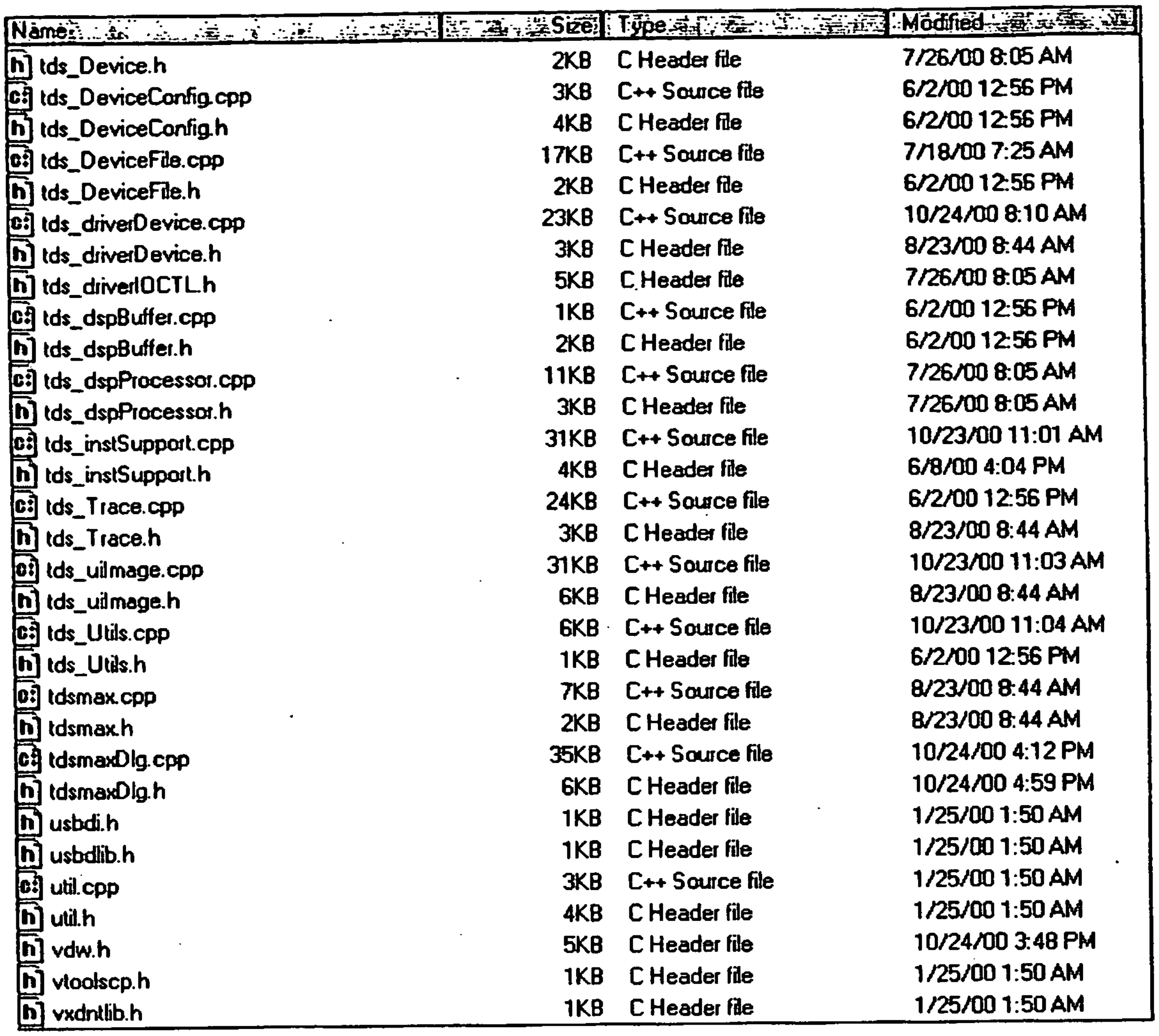
Name	Size	Type	Modified
khid.cpp	7KB	C++ Source file	1/25/00 1:50 AM
khid.h	8KB	C Header file	1/25/00 1:50 AM
khidaux.cpp	8KB	C++ Source file	1/25/00 1:50 AM
kicount.h	2KB	C Header file	1/25/00 1:50 AM
kimgsect.h	3KB	C Header file	1/25/00 1:50 AM
kintrupt.cpp	8KB	C++ Source file	1/25/00 1:50 AM
kintrupt.h	8KB	C Header file	1/25/00 1:50 AM
kiocparm.h	4KB	C Header file	1/25/00 1:50 AM
kirp.cpp	6KB	C++ Source file	1/25/00 1:50 AM
kirp.h	18KB	C Header file	1/25/00 1:50 AM
klist.cpp	18KB	C++ Source file	1/25/00 1:50 AM
klist.h	21KB	C Header file	1/25/00 1:50 AM
klower.cpp	10KB	C++ Source file	1/25/00 1:50 AM
klower.h	10KB	C Header file	1/25/00 1:50 AM
kmemory.h	7KB	C Header file	1/25/00 1:50 AM
kmutex.cpp	5KB	C++ Source file	1/25/00 1:50 AM
kmutex.h	4KB	C Header file	1/25/00 1:50 AM
kpcicfg.cpp	10KB	C++ Source file	1/25/00 1:50 AM
kpcicfg.h	9KB	C Header file	1/25/00 1:50 AM
kpnpdev.cpp	68KB	C++ Source file	1/25/00 1:50 AM
kpnpdev.h	18KB	C Header file	1/25/00 1:50 AM
kpnplow.cpp	5KB	C++ Source file	1/25/00 1:50 AM
kpnplow.h	7KB	C Header file	1/25/00 1:50 AM
kpownot.h	3KB	C Header file	1/25/00 1:50 AM
kquery.cpp	10KB	C++ Source file	1/25/00 1:50 AM
kquery.h	5KB	C Header file	1/25/00 1:50 AM
kregkey.cpp	23KB	C++ Source file	1/25/00 1:50 AM
kregkey.h	9KB	C Header file	1/25/00 1:50 AM
kresreq.cpp	31KB	C++ Source file	1/25/00 1:50 AM
kresreq.h	8KB	C Header file	1/25/00 1:50 AM
ks5920.h	17KB	C Header file	1/25/00 1:50 AM



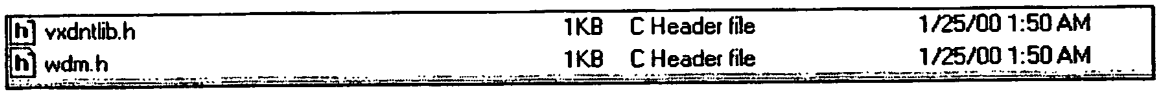
Name	Size	Type	Modified
ks5920.h	17KB	C Header file	1/25/00 1:50 AM
ks5933.h	58KB	C Header file	1/25/00 1:50 AM
ks59xx.h	45KB	C Header file	1/25/00 1:50 AM
ks59xxrd.h	12KB	C Header file	1/25/00 1:50 AM
ksemaphr.cpp	5KB	C++ Source file	1/25/00 1:50 AM
ksemaphr.h	4KB	C Header file	1/25/00 1:50 AM
ksfifo.cpp	3KB	C++ Source file	1/25/00 1:50 AM
ksfifo.h	2KB	C Header file	1/25/00 1:50 AM
kspin.cpp	2KB	C++ Source file	1/25/00 1:50 AM
kspin.h	4KB	C Header file	1/25/00 1:50 AM
kstdwmi.cpp	3KB	C++ Source file	1/25/00 1:50 AM
kstdwmi.h	2KB	C Header file	1/25/00 1:50 AM
Kstradpt.cpp	21KB	C++ Source file	1/25/00 1:50 AM
Kstream.cpp	29KB	C++ Source file	1/25/00 1:50 AM
Kstream.h	23KB	C Header file	1/25/00 1:50 AM
Kstrmdrv.cpp	16KB	C++ Source file	1/25/00 1:50 AM
ksysthrd.cpp	3KB	C++ Source file	1/25/00 1:50 AM
ksysthrd.h	5KB	C Header file	1/25/00 1:50 AM
ktimer.cpp	3KB	C++ Source file	1/25/00 1:50 AM
ktimer.h	4KB	C Header file	1/25/00 1:50 AM
ktrace.cpp	11KB	C++ Source file	1/25/00 1:50 AM
ktrace.h	11KB	C Header file	1/25/00 1:50 AM
kunitnam.cpp	4KB	C++ Source file	1/25/00 1:50 AM
kunitnam.h	3KB	C Header file	1/25/00 1:50 AM
kusb.cpp	88KB	C++ Source file	1/25/00 1:50 AM
kusb.h	44KB	C Header file	1/25/00 1:50 AM
kustring.cpp	9KB	C++ Source file	1/25/00 1:50 AM
kustring.h	6KB	C Header file	1/25/00 1:50 AM
kvxdintf.cpp	4KB	C++ Source file	1/25/00 1:50 AM
kvxdintf.h	8KB	C Header file	1/25/00 1:50 AM
kwmi.cpp	18KB	C++ Source file	1/25/00 1:50 AM



Name	Size	Type	Modified
kwmi.cpp	18KB	C++ Source file	1/25/00 1:50 AM
kwmi.h	47KB	C Header file	1/25/00 1:50 AM
kwmiblock.h	12KB	C Header file	1/25/00 1:50 AM
kwmistr.cpp	7KB	C++ Source file	1/25/00 1:50 AM
kwmistr.h	6KB	C Header file	1/25/00 1:50 AM
kworkitm.h	3KB	C Header file	1/25/00 1:50 AM
Myslider.cpp	2KB	C++ Source file	8/23/00 8:44 AM
Myslider.h	2KB	C Header file	8/23/00 8:44 AM
Ntddsnd.h	1KB	C Header file	6/2/00 12:56 PM
Ntddwave.h	5KB	C Header file	6/2/00 12:56 PM
OwnerDrawListBox.cpp	2KB	C++ Source file	8/23/00 8:44 AM
OwnerDrawListBox.h	2KB	C Header file	8/23/00 8:44 AM
PoolTag.cpp	2KB	C++ Source file	1/25/00 1:50 AM
PrefDlg.cpp	2KB	C++ Source file	8/23/00 8:44 AM
PrefDlg.h	2KB	C Header file	8/23/00 8:44 AM
PushButtons.h	3KB	C Header file	8/23/00 8:44 AM
resource.h	3KB	C Header file	8/23/00 8:44 AM
scm_Service.cpp	27KB	C++ Source file	10/23/00 10:54 AM
scm_Service.h	2KB	C Header file	8/11/00 3:36 PM
shfifo.cpp	3KB	C++ Source file	1/25/00 1:50 AM
shfifo.h	4KB	C Header file	1/25/00 1:50 AM
startdlg.cpp	2KB	C++ Source file	8/4/00 3:30 PM
startdlg.h	2KB	C Header file	8/4/00 3:30 PM
StdAfx.cpp	1KB	C++ Source file	6/2/00 12:56 PM
StdAfx.h	2KB	C Header file	8/23/00 8:44 AM
strmini.h	1KB	C Header file	1/25/00 1:50 AM
tds_AppBarIcon.cpp	6KB	C++ Source file	10/24/00 5:04 PM
tds_AppBarIcon.h	2KB	C Header file	6/2/00 12:56 PM
tds_counter.h	1KB	C Header file	7/26/00 8:05 AM
tds_Device.cpp	18KB	C++ Source file	10/23/00 10:58 AM
tds_Device.h	2KB	C Header file	7/26/00 8:05 AM



Name	Size	Type	Modified
tds_Device.h	2KB	C Header file	7/26/00 8:05 AM
tds_DeviceConfig.cpp	3KB	C++ Source file	6/2/00 12:56 PM
tds_DeviceConfig.h	4KB	C Header file	6/2/00 12:56 PM
tds_DeviceFile.cpp	17KB	C++ Source file	7/18/00 7:25 AM
tds_DeviceFile.h	2KB	C Header file	6/2/00 12:56 PM
tds_driverDevice.cpp	23KB	C++ Source file	10/24/00 8:10 AM
tds_driverDevice.h	3KB	C Header file	8/23/00 8:44 AM
tds_driverIOCTL.h	5KB	C Header file	7/26/00 8:05 AM
tds_dspBuffer.cpp	1KB	C++ Source file	6/2/00 12:56 PM
tds_dspBuffer.h	2KB	C Header file	6/2/00 12:56 PM
tds_dspProcessor.cpp	11KB	C++ Source file	7/26/00 8:05 AM
tds_dspProcessor.h	3KB	C Header file	7/26/00 8:05 AM
tds_instSupport.cpp	31KB	C++ Source file	10/23/00 11:01 AM
tds_instSupport.h	4KB	C Header file	6/8/00 4:04 PM
tds_Trace.cpp	24KB	C++ Source file	6/2/00 12:56 PM
tds_Trace.h	3KB	C Header file	8/23/00 8:44 AM
tds_uiImage.cpp	31KB	C++ Source file	10/23/00 11:03 AM
tds_uiImage.h	6KB	C Header file	8/23/00 8:44 AM
tds_Utils.cpp	6KB	C++ Source file	10/23/00 11:04 AM
tds_Utils.h	1KB	C Header file	6/2/00 12:56 PM
tdsmax.cpp	7KB	C++ Source file	8/23/00 8:44 AM
tdsmax.h	2KB	C Header file	8/23/00 8:44 AM
tdsmaxDlg.cpp	35KB	C++ Source file	10/24/00 4:12 PM
tdsmaxDlg.h	6KB	C Header file	10/24/00 4:59 PM
usbdi.h	1KB	C Header file	1/25/00 1:50 AM
usbdlib.h	1KB	C Header file	1/25/00 1:50 AM
util.cpp	3KB	C++ Source file	1/25/00 1:50 AM
util.h	4KB	C Header file	1/25/00 1:50 AM
vdw.h	5KB	C Header file	10/24/00 3:48 PM
vtoolscp.h	1KB	C Header file	1/25/00 1:50 AM
vxdntlib.h	1KB	C Header file	1/25/00 1:50 AM



vxdntlib.h	1KB	C Header file	1/25/00 1:50 AM
wdm.h	1KB	C Header file	1/25/00 1:50 AM

<u>Field of the Invention</u>

This application relates to an audio device driver, in particular, to an audio device driver that includes a component driver capable of intercepting a signal pointing to stored digitized audio information such that the digitized information is subjected to sound modification processing and/or stored in memory prior to being received by a sound playback system and, more particularly to such a driver that operates in kernel mode.

Background of the Invention

Computer operating systems manage the operation of multiple application programs concurrently. Such programs typically write (issue) digitally represented sound or audio data to a sound playback system. "Writing" digitized audio data to a sound playback system involves the application program storing the digitized data in computer memory that it controls as well as generating an I/O request, typically a WaveOut signal, to the sound playback system, wherein the request includes pointers to the stored digitized audio data. The operating system retrieves the digitized audio data from the initial memory locations and stores the data in new memory locations it controls and changes the pointers in the I/O request to reflect the new memory locations. A sound playback system is defined herein as a sound card and one or more speakers. The computer operating system receives the I/O request originating from the application program and outputs a modified request in a format acceptable to the sound card, which request may point to the stored digitized audio information. The sound card then retrieves the digitized audio information from memory and converts the information into an analog signal, which is used to drive the one or more speakers. Digitized audio information may be read or retrieved from one or more of a plurality of audio sources, including compact disks (CDs), digital video disk (DVD) players, computer games, files (e.g., MP3 files) downloaded over the Internet, and streaming audio received over the Internet. Special considerations attach to the use of each of these audio sources.

Compact Disk Playback

Playback of music recorded on a compact disk requires a compact disk player and an application software program for causing the compact disk player to scan the compact disk and read digital information stored thereon. Digital information representing audio information is then written by the application program to the sound playback system.

DVD Playback

Requires a DVD player and an application software program for causing the DVD player to read digital information representing both audio and video information. Audio information is separated from the video data and handled in a manner similar to compact disk playback.

Computer Game Playback

Playback of sounds related to computer games requires that the user execute game application software. At various times determined by the game application software, it writes digital information representing sounds to the sound playback system.

Musical Files Retrieved from the Internet

Once music files have been retrieved from the Internet, they may remain in computer memory, stored on magnetic disks or stored onto compact disks. Playback of files retrieved from the Internet requires a special player that can locate the files from computer memory and/or magnetic disk and/or compact disk. The special player then reads the digital information representing sound information from the stored location and writes it to a sound playback system.

Streaming Audio Retrieved from the Internet

Playback of streaming audio information retrieved from the Internet requires a special player. The special player must retrieve real-time audio streams of digital information from the Internet. The special player must then write these streams of digital data representing audio information to a sound playback system.

Each distinct sound application program (e.g., CD or DVD player program, computer game program, retrieved musical file program, retrieved streaming audio program) has the capability to effect the reading or retrieving of digital information from an appropriate source (e.g., CD or DVD player, etc.) and write that digital information to a sound playback system. The application programs typically operate in "user mode." A computer typically has several modes of operation relative to the notion of "privilege." Privilege has to do with the degree of access that a program has to various functional capabilities and resources contained within a computer. "User Mode" is typically a designated mode wherein the current executing program has the most limited access to computer resources and functions. Typical user application programs operate in "user mode." In contrast, "Kernel Mode" is typically a privileged mode wherein the current executing program has complete and unlimited access to all computer functional capabilities and resources. Only operating system components have access to kernel mode.

In a typical modern computer, because more than one user application program may be executing concurrently, user programs are not permitted direct access to shared resources, including but not limited to sound playback systems. Shared resources are resources typically shared across a multiplicity of application programs. Typical shared resources include but are not limited to: (1) monitor, (2) disk drives, (3) sound cards, (4) printers. Therefore, when an application program "writes" to a sound playback system, the operating system intervenes, intercepting the signal pointing to the stored digital information representing the sound to be played. It also transfers the digitized audio data to new memory locations and modifies the signal so as to point to the new memory locations containing the digitized audio data. The operating system then transmits the signal to the sound playback system on behalf of the application program. This intervention prevents sounds emanating from various application programs from becoming intermixed and therefore confused during playback. This intervention also permits various application programs to use common commands with which to "write" digital sound information to the wide variety of available sound playback systems.

The quality of the sound produced by each of the above mentioned audio application programs can be improved by applying sound modification processing. In each case, a listener's sound experience can be enhanced or otherwise altered by appropriate digital sound modification processing (e.g., equalization, volume control, 3-dimensional processing, reverberation, etc.). The motivation for such modifications may vary from application to application, and the particular modifications may vary as well. However, the requirement and desirability of modifications is common to all such applications.

Previously, some sound cards have been provided with equalization hardware so as to effect a type of sound modification. However, the sophistication level of the sound processing hardware provided on sound cards is typically kept low. This is because sound processing hardware capable of effecting higher levels of equalization is costly and consumes valuable sound card real estate. Some application software programs are provided with sound modification processing capabilities. However, the manner of sound modification processing and its effectiveness varies from software package to software package. Hence, consistent modification of the digital audio information written by a plurality of application software packages typically cannot be achieved. Accordingly, there is a need for a system which is capable of capturing more, and up to all, digital audio data written to a sound playback system by a number of application programs such that the data can be modified in the same or a consistent manner prior to the data reaching the sound playback system.

Summary of Invention

In accordance with the present invention, an audio device driver is provided including a component driver capable of intercepting a signal pointing to digitized audio information or data, wherein both the signal and the audio information originate at an application program, such that the stored digitized audio information can be modified and/or stored elsewhere before being provided to a sound playback system. Preferably, the digitized audio information or data being pointed to is stored. The intercepting component driver may include software for modifying the stored digitized data via 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and/or volume control and/or software for storing the digitized data elsewhere. Alternatively, the intercepting component driver may transfer the signal to a separate software module which is capable of modifying the stored digitized data via 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and/or volume control, and/or software for storing the digitized audio data elsewhere. After the digitized audio data is modified and/or stored, the signal pointing to the audio data is received by a functional component driver, which provides the signal to a sound card in a format acceptable to the sound card. The sound card then retrieves the audio data from memory and converts the data into an analog signal, which is used to drive one or more speakers.

The intercepting component driver comprises an executable program stored in computer memory. The executable program is in a logical data flow path between one or more application programs and one or more sound playback systems, i.e., the intercepting component driver could be incorporated into two or more device drivers associated with a like number of sound cards. The intercepting driver executes in kernel mode to attain access to shared resources, e.g., a sound playback system not directly accessible to application software programs. As such, the intercepting driver is a natural operating system extension since it operates in a kernel, privileged mode and therefore has access to any and all computer functions and resources, including the sound playback system.

Processing of digitized audio information or data is dependent upon several parameters of the data, subject to the constraint that sufficient processing time is available. The parameters of primary interest include: (1) Sampling rate, (2) Bits per sample, (3) Number of channels, (4) Code format.

<u>Sampling Rate:</u> The sampling rate indicates the number of bits recorded for each sample of the digital information. For example, a musical Compact Disk usually contains sampled data at 44,100 samples per second. Other sampling rates typically available are: (1) 22,050 samples per second, (2) 11,025 samples per second, (3) 48,000 sample per second. Many other sampling rates exist. Processing of the digital information is dependent upon the sampling rate. The intercepting component driver deduces the number of samples per second and processes the information in accordance with the number of samples per second contain within the digitized information.

<u>Bits per sample:</u> The bits per sample indicate the accuracy of the samples and the dynamic range available. For example, compact disk music is sampled at a resolution of 16 bits per sample. Furthermore, the compact disk music represents digital sound information in a two's complement -- +32,767 to -32,678 -- so that there is a total range of 65,535 or about 96 dB of range. Processing of the digital sound information is dependent upon the number of bits per sample. Therefore, the intercepting component driver must deduce the number of bits per sample in the represented digital information and process the information accordingly. Some data is recorded at 8 bits per sample. The intercepting driver must extract and process the eight bits per sample data differently than the sixteen bits per sample noted above.

<u>Number of Channels:</u> Each digital sound record contains the number of channels. The data could be monophonic (one channel of digital sound information), stereophonic (two channels of information), quadraphonic (four channels of information). The information may be in any number of channels. Typically, the digital sound data is

located in a memory buffer in sequential locations. Data from the various channels is typically interleaved in successive locations in the sound buffer. Since each channel of information is processed independently, the intercepting component driver must deduce the number of channels represented by the digital sound information. Each distinct channel is then processed independently.

Code Format: The digital sound information may be recorded in several code formats. For example, as discussed above, compact disk music information is typically recorded in two's complement format. This means that, in the case of compact disk music, the positive data is represented as pure binary numbers (0 through 32,767) and negative numbers are represented as two's complement numbers (-32,768 through -1). This so-called 16-bit pulse code modulation (PCM) format is a usual but not necessarily invariant format for 16-bit information. In some cases, the code format differs from the 16 bit two's complement PCM format discussed above. Some computer sounds are recorded in 8 bit per sample excess 128 format. In this case the data is represented as 8 bits of PCM code for each digital data sample. The excess 128 format means that the data is stored as standard binary information except that the digital code for 128_{10} represents zero. All PCM data above 128 (129 to 255) represents positive data and all PCM data below 128 (0 to 127) represents negative data. The processing of sound data is dependent upon the exact code in which the data is represented. Therefore, the intercepting component driver must deduce the particular code format in which the data is represented prior to processing the sound data.

Once the intercepting component driver has deduced all of the parametric information, set out above, the driver may initiate or invoke processing software which may be incorporated into the intercepting component driver or comprise a separate software module. The intercepting driver software or the software module executes a digital software modification or processing function, such as 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; volume control; and/or data memory storage. The intercepting component driver or separate processing module stores the modified and/or stored information in a buffer and subsequently passes a signal pointing to the processed data to the next component driver in the device driver.

The intercepting driver software or the software module may comprise a digital filter for effecting sound enhancement processing. The digital filter enhances the harmonic quality of digitized audio data, in particular audio data having a complex wave form (i.e., multiple frequency components such as, for example, music, singing, speech, animal sounds, naturally occurring sounds, equipment noises, and the like), such that the digitized audio data enhanced via the filter exhibits an improved harmonic quality compared to that of the input digitized audio data. It has been found that a similar harmonic enhancement can be obtained using the circuits disclosed in U.S. Patent No. 5,361,306; U.S. Patent Application Serial No. 08/472,876, having a filing date of June 7, 1995 and entitled APPARATUS AND METHOD OF ENHANCING ELECTRONIC AUDIO SIGNALS; U.S. Patent Application Serial No. 08/700,728, having a filing date of August 13, 1996 and entitled APPARATUS AND METHODS FOR THE HARMONIC ENHANCEMENT OF ELECTRONIC AUDIO SIGNALS;U.S. Patent Application Serial No. 08/909,807, having a filing date of August 12, 1997 and entitled APPARATUS AND METHODS FOR THE HARMONIC ENHANCEMENT OF ELECTRONIC AUDIO SIGNALS; U.S. Patent Application Serial No. 08/989,373, having a filing date of December 12, 1997 and entitled APPARATUS AND METHODS FOR ENHANCING ELECTRONIC AUDIO SIGNALS; U.S. Patent Application Serial No. 09/431,371, having a filing date of November 1, 1999 and entitled DIGITAL FILTER; and PCT Patent Application, Serial No. PCT/US00/33197, entitled "Apparatus and Methods for Enhancing Electronic Audio Signals," filed on December 7, 2000, the disclosures of which are incorporated herein by reference.

In accordance with a first aspect of the present invention, a device driver is provided forming part of or an extension of an operating system. The operating system is adapted to receive digitized audio data from an application program. The device driver comprises: an intercepting component driver which is capable of initiating at least one of sound modification processing of the digitized audio data and storage of the digitized audio data.

In one embodiment, the component driver effects sound modification processing of the digitized audio data. The sound modification processing may comprise at least one of 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and volume control.

The sound modification processing may comprise sound enhancement processing using a digital filter. The digital filter may comprise a series of digitized time coefficients stored in a memory. The time coefficients are mapped to a like number of frequency coefficients. The frequency coefficients are spaced at frequency intervals having either zero phase angles or linearly spaced phase angles and having amplitudes which are mirrored about a mid frequency to produce periodicity of a time response for the digital filter.

The time coefficients are odd in number, and the number of the time coefficients may be equal to or greater than 5. It is also contemplated that the time coefficients may be even in number.

The time coefficients may be defined by inverse discrete Fourier transforms of the frequency coefficients.

A portion of the frequency coefficients having frequencies within a free band may be selected so as to achieve a generally constant oscillation frequency across a center band which is broader than the free band.

The frequency coefficients may be spaced at equal frequency intervals.

The time coefficients may be integer numbers.

In addition to effecting sound modification processing, the component driver may further initiate storage of the digitized audio data on one of a compact disk, a digital video disk, a floppy disk and computer memory.

Alternatively, the component driver may initiate storage of the digitized audio data on one of a compact disk, a digital video disk, a floppy disk and computer memory without effecting sound modification processing of the digitized audio data.

The intercepting component driver is capable of initiating the at least one of sound modification processing of the digitized audio data and storage of the digitized audio data in kernel mode.

In accordance with a second aspect of the present invention, a device driver is provided forming part or an extension of an operating system. The operating system is adapted to receive digitized audio data from an application program. The device driver comprises: at least two layered component drivers including a functional component driver which manages the operation of a sound card device and an intercepting component driver positioned above the functional driver and being capable of initiating at least one of sound modification processing of the digitized audio data and storage of the digitized audio data.

The digitized audio data may be stored in a first portion of computer memory and the intercepting component driver may initiate the at least one of sound modification processing and storage by intercepting a signal pointing to the digitized audio data stored in the first portion of computer memory and output by the application program, retrieving the digitized data from the first portion of memory and effecting the at least one of sound modification processing of the digitized data and storage of the digitized data.

In one embodiment, the intercepting component driver effects sound modification processing of the digitized data. The component driver may effect the sound modification of the digitized audio data via at least one of 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and volume control.

The intercepting component driver may effect sound enhancement processing of the digitized audio data using a digital filter. The digital filter comprising a series of digitized time coefficients stored in a memory. The time coefficients are mapped to a like number of frequency coefficients. The frequency coefficients are spaced at frequency intervals having either zero phase angles or linearly spaced phase angles and having amplitudes which are mirrored about a mid frequency to produce periodicity of a time response for the digital filter. It is also contemplated that non-linear spaced phase angles may be employed.

In addition to effecting sound modification processing, the intercepting component driver may further cause the digitized audio data to be stored on one of a compact disk, a digital video disk, a floppy disk and a second portion of computer memory.

In another embodiment, the intercepting component driver may cause the digitized audio data to be stored on one of a compact disk, a digital video disk, a floppy disk and a second portion of computer memory without effecting sound modification processing.

In accordance with a further embodiment of the present invention, the digitized audio data is stored in a first portion of computer memory and the intercepting component driver initiates the at least one of sound modification processing and storage by intercepting a signal pointing to the digitized audio data stored in computer memory and output by the application program, transferring the signal to a separate software module such that the module may retrieve the digitized data from the first portion of memory and effect the at least one of sound modification processing of the digitized data and storage of the digitized data.

In one embodiment, the software module effects sound modification processing of the digitized data. The software module may effect the sound modification of the digitized audio data via at least one of 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and volume control.

The software module may effect sound enhancement processing of the digitized audio data using a digital filter. The digital filter comprises a series of digitized time coefficients stored in a memory. The time coefficients are mapped to a like number of frequency coefficients. The frequency coefficients are spaced at frequency intervals having either zero phase angles or linearly spaced phase angles and having amplitudes which are mirrored about a mid frequency to produce periodicity of a time response for the digital filter. It is also contemplated that non-linear spaced phase angles may be employed.

The software module may further cause the digitized audio data to be stored on one of a compact disk, a digital video disk, a floppy disk and a second portion of computer memory.

In another embodiment, the software module causes the digitized audio data to be stored on one of a compact disk, a digital video disk, a floppy disk and a second portion of computer memory without effecting sound modification processing.

The intercepting component driver is capable of initiating the at least one of sound modification processing of the digitized audio data and storage of the digitized audio data in kernel mode.

In accordance with a third aspect of the present invention, a computer-implemented method is provided for creating processed sound from digitized audio data. The process comprises the steps of: storing the digitized audio data in memory; creating a kernel mode streaming signal pointing to the stored audio data; using the kernel mode streaming signal for kernel mode enhancement of the data, the kernel mode enhancement comprising real time inverse convolution of the audio data and a series of time coefficients derived from frequency coefficients inversely related to the response of the human ear across a range of frequencies; and using the audio data, enhanced as aforesaid, for kernel mode generation of processed sound.

The enhancement of the audio data may be performed by an intercepting component driver in a device driver or a separate software module.

In accordance with a fourth aspect of the present invention, a method is provided for processing sound comprising the steps of: 1. generating a set of frequency coefficients inversely related to the responsiveness of the human ear, 2. converting the frequency coefficients to time coefficients, 3. sampling the sound to generate an endless series of raw audio data, 4. storing the raw audio data, 5. generating a series of addresses indicating the locations of the stored data, 6. reading the raw audio data from memory, 7. generating a series of processed audio data by convolving the raw audio samples against the time coefficients, and 8. using the processed audio data to drive a sound generator, such as one or more speakers.

The steps 4 through 7 may be performed by a digital computer operating in kernel mode.

Brief Description of the Drawings

Fig. 1 is a block diagram illustrating components of a conventional computer operating system having layered device drivers;

Fig. 1A illustrates a device control block and call-down table of a prior art device driver;

Fig. 2 illustrates a device driver constructed in accordance with the present invention along with an API and an input output system;

Fig. 3 illustrates K-mode structures for KS-Property IRP evaluation;

Fig. 4 illustrates K-mode structures for KS_Write_Stream IRP evaluation;

Figs. 5-7 are illustrations of frequency response curves for a digital filter configured in accordance with first, second and third embodiments, respectively, of the present invention; and

Fig. 8 is a block diagram of an intercepting component driver constructed in accordance with the present invention.

Description of the Preferred Embodiments

The present invention contemplates both methods and systems for dynamically extending an operating system, which uses one or more layered device drivers, each comprising a plurality of data managers or component drivers, in fulfilling I/O requests issued by an application program or software, such as an audio application program. More specifically, the present invention contemplates interrupting the normal sequence of processing in the operating system such that an audio I/O request, which includes pointers to stored digitally represented sound information issued by an application program to a sound playback system, can be intercepted by a component driver. The component driver then initiates sound modification processing of the digitized audio data and/or storage of the digitized data elsewhere in computer memory or on a separate memory device before the data is provided to a functional or lower-most component driver. The intercepting component driver may include software for effecting sound modification processing such as 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and/or volume control. Alternatively, the intercepting component driver may transfer the I/O request to a separate software module having the capability to effect sound modification processing such as 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and/or volume control. After the digitized audio information is modified and/or stored, the request pointing to the audio information is received by a functional component driver, which provides the request to a sound card in a format acceptable to the sound card. The sound card then retrieves the audio data from memory and converts the data into an analog signal, which is used to drive one or more speakers.

Fig. 1 is a block diagram illustrating components of a conventional operating system 201 having device drivers configured during computer start-up or restart. The system 201, once configured, functions as follows. An application program 200 formulates an input/output (I/O) request and passes that request to the operating system 201 through an application program interface 202. The application program interface 202 provides the application program 200 with access to the operating system 201. When the application program interface 202 receives the I/O request, it invokes an input-output system (IOS) 203, passing it the request. The input-output system 203 determines which of a plurality of device drivers, only device drivers 204 and 205 are illustrated, is identified in the request. Device driver 204 comprises a plurality of component drivers 210, 211, ***, N Driver, while device driver 205 comprises a plurality of component drivers 220, 221, ***, N Driver. Component drivers 210 and 220 comprise upper-most drivers, while component drivers N Drivers, also referred to herein as functional drivers, are the lower-most drivers.

Assuming device driver 204 is the driver identified in the request, the input-output system 13 initially invokes the upper-most layer component driver 210, passing it the request. The upper-most layer driver 210 performs component driver specific functions and invokes, when appropriate, the next lower layer component driver 211, passing it the request. The next lower layer component driver performs its own component driver specific functions and invokes, when appropriate, the next lower layer component driver, passing it the request, and so on. Finally, the lower-most driver N Driver interacts with a device adapter, such as a device adapter for a disk drive.

Fig. 1A is a block diagram illustrating conventional data structures that support the layering of component drivers in a device driver. While the data structures for each device driver are identical, those illustrated in Fig. 1A will be described as corresponding to the device driver 204. The device driver 204 contains a device control block (DCB) 402, a call-down table (CDT) 403, and a plurality of component drivers 210, 211, ***, N Driver. As is illustrated in Fig. 1, device driver 204 corresponds to device 208. The device control block 402 contains information indicating the type of the device 208 (disk drive, etc.), and a pointer 43A that will point to each entry 403A, 403B, ***, 403N in the call-down table 403 so as to invoke the component driver identified by the entry. The call-down table entries 403A, 403B, ***, 403N also have pointers 404A, 404B, ***, 404N, which point to the component drivers 210, 211, ***, N Driver of the device driver 204. The call-down table 403 specifies the order in which the component drivers are executed to process the I/O request. The upper-most (first) entry 403A in the call-down table 403 points to the upper-most (first) layer component driver 210. The second entry 403B in the call-down table 403 points to the second layer component driver 211, and so on. Each layer component driver invokes the next lower layer component driver. Thus, once the IOS 13 invokes the upper-most layer component driver 210, the layer component drivers control the execution of the device driver 204.

When the IOS 203 receives an I/O request, it selects the device control block 402 of the device driver 204 or 205, which is to process the I/O request. It then stores in the selected control block 402 an entry pointer 43A to the first entry 403A of the call-down table 403 corresponding to the selected device control block 402. The IOS 13 invokes the component driver pointed to by the first entry 403A in the call-down table 403 related to the selected device control block 402, passing the component driver the selected device control block 402. Generally, the component driver performs its component driver specific functions, adjusts the entry pointer 43A in the device control block 402 to point to the entry in the call-down table 403 for the next component driver so as to invoke the next lower component driver. The next component driver performs its own component driver specific functions and invokes, when appropriate, the next lower layer component driver, and so on.

At computer startup or restart, the operating system 201 invokes a driver configuration routine that is part of the IOS 13. The driver configuration routine inputs (typically from a storage device) a load table (not shown) and creates and initializes a device control block and a call-down table for each device. The load table contains a list of pointers to component drivers that are available to be configured into device drivers. The driver configuration routine scans the load table and loads into memory each component driver in order, starting with the component driver pointed to by the last load table entry. A component driver is selected for inclusion into a device driver for a particular device by allowing the component driver itself to determine whether to be included. Once a component driver is loaded, the configuration routine invokes the component driver to determine whether it should be included in the call-down table for the device driver. If the component driver determines that it should be included, then the component driver includes itself into the call-down table for that device driver by inserting a pointer to itself in the call-down table. The call-down table functions as a stack in that new entries are pushed onto the top of the call-down table and existing entries are pushed down. When multiple device drivers are to be configured, the configuration routine invokes each component driver once for each device driver, in the fashion described above.

Fig. 2 illustrates an audio device driver 300 constructed in accordance with the present invention. It forms part of a modified computer operating system. An operating system, capable of being modified in accordance with the present invention, may comprise a commercially available 32-bit operating system (OS) developed by Microsoft Corporation and distributed under the trademarks Windows 98, Windows NT and Windows 2000. Many Windows operating systems support layered device drivers. Such layered device drivers are described, for example, in U.S. Patent Nos. 5,613,123; 5,781,797; 5,931,935, each assigned to Microsoft Corporation, the disclosures of which are hereby incorporated herein. Typically, layered device drivers are designed in accordance with a Windows Driver Model (WDM) specification, created by Mircosoft Corporation. Further information regarding the WDM architecture and layered device drivers is set out in the following documents, each of which is incorporated herein by reference: Windows 2000 Driver Development Kit, Microsoft Press, 2000; Programming the Microsoft Windows Driver Model, Walter Oney, Microsoft Press, 1999; Windows NT Device Driver Development, P.G. Viscarola and W. G. Mason, Macmillan Technical Publishing, 1999; Writing Windows WDM Device Drivers, Chris Cant, R&D Books, 1999; Windows 98 Developer's Handbook, ben Ezzell with Jim Blaney, Sybex, Inc., 1998; The Windows Device Driver Book, Art Baker, Prentice Hall, PTR, 1997; and Windows NT File System Internals, A Developer's Guide, O'Reilly & Associates, 1997. The audio device driver 300 comprises a layered device driver.

Referring again to Fig. 2, an audio source application program, such as a media player 10, interacts with a hardware device, e.g., a compact disk player, a DVD player or like device, so as to retrieve digitized audio information or data from a memory device, e.g., a compact disk (CD), a digital video disk (DVD), and then stores the digitized audio data in memory buffers it controls in computer memory. Alternatively, the media player 10 retrieves digitized audio information from files (e.g., MP3 files) stored in computer memory, or processes digitized audio information generated by a computer game or received over the Internet as streaming audio and subsequently stores the digitized audio data in memory buffers. The digitized audio data comprises pulse code modulated (PCM) data representing sound. The media player 10 then issues an I/O request in the form of a WaveOut command or signal, which is received by an application program interface (API) 12. The request contains pointers to the buffers in computer memory containing the digitized audio information. The API 12, in turn, modifies the WaveOut command so that it is in a format acceptable to an input output system (IOS) 13, which receives the command.

The IOS 13 forms part of the computer operating system. It determines which device driver can service the request based on the device driver identified in the WaveOut command. Assuming that the IOS 13 determines that the I/O request has identified device driver 300 corresponding to sound card 42, illustrated in Fig. 2, the IOS 13 first converts the WaveOut command into I/O request packets or IRPs, as will be described below, and passes those IRPs to the device driver 300.

After receiving the WaveOut command from the media player 10, the API 12 transfers the WaveOut command to an audio processing program 14, one of which was developed by Microsoft and is known as "winMM.DLL." The audio processing program 14 transfers the digitized audio data into other buffers in computer memory, which buffers are controlled by the operating system. It also changes the pointers in the WaveOut command to reflect the new memory locations. It then relays the WaveOut command to a WDM audio driver 18, which also operates in the user mode, and may comprise a Microsoft driver known as "WDMAUD.DRV." The driver 18 translates the WaveOut command into IOCTL messages, which are provided to a kernel mode WDM audio driver 22. Audio driver 22 may be a Microsoft file termed "WDMAUD.SYS." The driver 22 converts IOCTL messages into packets of information that are formatted correctly for processing inside the kernel mode. Those packets of information are generally referred to as I/O request packets or IRPs. The IRPs contain pointers to the buffers in computer memory containing the digitized audio data. The IRPs are provided to system audio device 26 and kernel mixer and sample rate converter 30, before being provided to the device driver 300. Files winMM.DLL; WDMAUD.DRV; and WDMAUD.SYS are incorporated into most Windows operating systems.

System audio device 26 may comprise Microsoft file "SYSAUDIO.SYS," which is also incorporated into most Windows operating systems. Sample rate converter 30 may comprise Microsoft file "KMIXER.SYS," also incorporated into most Windows operating systems, which provides system-wide audio mixing, bit-depth conversion, and sample rate conversion for PCM audio streams. All PCM data may be converted to 44.1 KHz, 16-bits samples.

As set out above, the kernel mixer and sample rate converter 30 passes IRPs to the device driver 300. More specifically, those IRPs are received by an intercepting component driver 50 which, in the illustrated embodiment, is the upper-most component driver in the device driver 300. It is noted that another component driver, which performs a different function from that of the component driver 50, may be positioned above component driver 50 in the call-down table for the device 42.

The component driver 50, when activated via a user program 55, captures IRPs (pointing to stored digitized audio data or samples) streaming through the component device 50 and sends those IRPs directly to one or more processing modules 51. The one or more processing modules 51 retrieve the data from first computer memory locations, modify the digitized audio signal via one of 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and/or volume control, and store the modified digitized audio data back in the first computer memory locations. Alternatively, the modified digitized audio data could be stored in second computer memory locations. For example, a first software module 51 may be provided for effecting a conventional 3-dimensional processing scheme while a second module 51, placed in series with the first module, may be provided for effecting volume control. Alternatively, a single processing module 51 could be provided for effecting those two functions. The final processing module 51 directs the IRPs pointing to the modified digitized audio data back to the component driver 50 so as to be sent by the driver 50 to the next component driver in the call-down table, which, in the illustrated embodiment, is port class component driver 34. Alternatively, the final processing module 51 may divert the IRP's pointing to the modified data directly to the port class driver 34.

It is also contemplated that the component driver 50 may divert the IRPs pointing to the modified digitized audio data to a storage module 52A, i.e., a storage device driver, so that the processed data may be stored on a storage medium such as computer memory (other locations), a DVD, a CD, a floppy disk or the like. The IRPs pointing to the modified and stored digitized audio data are then sent back by the storage module 52A to the component driver 50 so as to be sent by the driver 50 to the next component driver in the call-down table, which, in the illustrated embodiment, is port class component driver 34. Alternatively, the storage module 52A may divert the IRP's pointing to the modified and stored data directly to the port class driver 34.

It is further contemplated that the component driver 50 may transfer the IRPs pointing to the digitized audio data directly to a storage module 52 without ever sending the IRPs to a processing module 51. The IRPs pointing to the stored data are then transferred either back to the component driver 50 or directly to the port class component driver 34.

The port class component driver 34 comprises a kernel mode dynamic link library (DLL) that converts IRPs into streams of data compatible with Industry Standard Architecture (ISA), direct memory access (DMA) or peripheral connection interface (PCI) audio device drivers. Those streams of data are passed to an audio adapter component driver 38, comprising a functional driver, which interfaces directly with the sound card 42. The driver 38 is typically provided by the hardware manufacturer and acts to convert the streams of data from the driver 34 into a format acceptable to the sound card 42. The sound card 42 then retrieves the modified and/or stored digitized audio data from memory (or the audio adapter component driver 38 retrieves the modified and/or stored digitized audio data from memory and then writes the data to the sound card 42) and converts the data into an analog signal for driving one or more speakers (not shown).

It is contemplated that any number of additional component drivers (not shown) may be positioned directly or indirectly above or below the intercepting component driver 50.

The installer software for the component driver 50 adds an "UpperFilters" entry into the operating system registry. This operation is performed as follows.

The installation software locates all eligible audio devices (i.e., sound cards) on the system and presents a list of the audio devices to a user for selection of the device driver to receive component driver 50. To locate all eligible audio devices, the installation software executes the following steps.

1. Create a list of all subkeys under "system\CurrentControlSet\control\DeviceClasses\KSCATEGORY_AUDIO" in the registry. The DeviceClasses key contains information about the device interfaces on the machine. The subkeys under this key represent each instance of an interface registered for the particular class interface. The device class interface we are interested in is the KSCATEGORY_AUDIO interface represented by the GUID 69941D04-93EF-11D0-A3CC-00A0c9223196.
2. The subkey for each registered instance is checked to verify that it contains an entry for DeviceInstance, and RefCount.
3. The subkey for each registered instance is checked to see that it contains a subkey named ?WAVE. (? Represents a wildcard)
4. The DeviceInstance key under "System\CurrentControlSet\Enum" (this is the registry path for win2k ..."Enum" Key may have a different path on other OS's) to make sure it contains a ClassGuid value equal to KSWAVEOUT_GUID_KEY.

If a candidate device (i.e., a sound card) passes the above checks it is added to a list for user selection. The user then selects an audio device from the list of eligible candidates. An "UpperFilters" entry specifying the component driver 50 is added/modified under the DeviceInstance subkey in the "ENUM" branch of the registry for the selected device.

The "UpperFilters" entry comprises a pointer in the operating system registry. The operating system configures a load table from the entries contained in the registry. In particular, the operating system first places all functional component drivers in the load table and then inserts all "upper filter" component drivers in the load table.

As discussed above, a driver configuration routine scans the load table and loads into memory each component driver in order, starting with the component driver pointed to by the last load table entry. A component driver is selected for inclusion into a device driver for a particular device by allowing the component driver itself to determine whether to be included. The component driver 50, once it determines it should be part of the device driver 300, is appropriately placed in the call-down table for the device driver 300 by issuing the following program instructions:

```
PDEVICE_OBJECT
  IoAttachDeviceToDeviceStack(IN PDEVICE_OBJECT SourceDevice, IN
PDEVICE_OBJECT TargetDevice);
```

These instructions, which are issued by any "upper filter" component driver so as to be appropriately positioned in a corresponding call-down table, are taught in a Microsoft publication entitled "Microsoft Device Driver SDK," which is part of a subscription available from Microsoft Corporation entitled "Microsoft Developer Network," (MSDN), the disclosure of which is hereby incorporated by reference.

Because the component driver 50 is positioned in the call-down table above the audio adapter component driver 38, all IRPs directed toward that component driver 38, and, hence, the sound card 42, first pass through the component driver 50. The component driver 50 is first only interested in Device_Control IRPs; all others are passed along to the port class component driver 34 below. Of the Device_Contol IRPs, the component driver 50 is interested in identifying two specific types: 1) KS_PROPERTY 2) KS_WRITE_STREAM. The KS_PROPERTY IRP precedes the KS_WRITE_STREAM IRP and provides such format information as a sampling rate, number of channels, etc. The KS_WRITE_STREAM IRP provides a pointer to the buffer in computer memory which actually contains the PCM wave data or digitized audio data to be processed.

The determination as to the major classification of the IRP (Device_Control, Write, Close, etc.) is made by looking at the MAJORFUNCTION member of the IO STACK structure, which is part of the IRP. Once it is determined that the IRP is a Device_Control IRP, the sub-type (KS_PROPERTY, KS_WRITE_STREAM) is determined by evaluating the IOCODE member of the DeviceControl structure. The DeviceControl structure is a member of the Parameters structure contained in the IO STACK.

Once the component driver 50 finds a KS_Property IRP, it examines two buffers that are associated with the IRP, see Fig. 3. The Parameters.DeviceControl.IoctlType3InputBuffer (A pointer to the buffer is contained in the subject IRP) is first cast as a KSPROPERTY structure. The component driver 50 then examines the members of this structure/buffer to determine if this is the IRP of interest. The Set member of KSPROPERTY, a GUID identifying the property set, is verified to conform to STATIC_KSPROPSETID_Connection. Next, the ID member of KSPROPERTY is verified to conform to KSPROPERTY_CONNECTION_DATAFORMAT. Finally the component driver 50 verifies that the Flags member of KSPROPERTY conforms to KSPROPERTY_TYPE_SET. Assuming that the first buffer examined conforms to the format and values described above, the component driver 50 examines the second buffer pointed to by the UserBuffer field of the KS_PROPERTY IRP.

A GUID is an abbreviation for Globally Unique Identifier. These identifiers are known to be unique over all space and a very large span of time. The Open Software Foundation (OSF) formulates these identifiers. The industry standard GUIDs are stored in Microsoft header files termed "ksmedia.h" and "ks.h".

The second buffer is cast as a KSDATAFORMAT_WAVEFORMATEX data structure. The KSDATAFORMAT_WAVEFORMATEX structure/buffer consist of two structures: namely (1) KSDATAFORMAT and (2) WAVEFORMATEX. The component driver 50 first looks at the members of the KSDATAFORMAT structure. The MajorFormat member, a GUID specifying the general format type, is verified to conform to STATIC_KSDATAFORMAT_TYPE_AUDIO. Next the SubFormat member, a GUID specifying a general sub-format, is verified to conform to: STATIC_KSDATAFORMAT_SUBTYPE_PCM. Finally the Specifier member, a GUID that provides additional data format type information, is verified to conform to

STATIC_KSDATAFORMAT_SPECIFIER_WAVEFORM. If all of the members verified above are correct, the component driver 50 then extracts information from the WAVEFORMATEX structure. The data to extracted from the WAVEFORMATEX structure are: (1) the number of channels (e.g., 2 for stereo, 1 for mono, etc.) (2) Sampling rate of the PCM data in samples per second, (3) the transfer rate of PCM data in bytes per second, (4) the sampling resolution in bits per sample, (5) the block alignment parameter, (6) a format tag. The member fields of this structure are named as follows: nChannels, nSamplesPerSec, nAvgBytesPerSec, wFormatTag, wBitsPersample, and nBlockAlign.

This completes the first major portion of the processing of the component driver 50.

Once the component driver 50 finds a valid KS_Property IRP, it looks for a KS_WRITE_STREAM IRP referencing valid PCM wave data. The User Buffer field of the IRP points to the buffer of interest. This buffer is first cast as a KSSTREAM_HEADER structure, see Fig. 4. The component driver then makes some safety checks. The following fields are examined. The fields are: (1) Data, (2) FrameExtent, (3) DataUsed. The Data field contains a pointer to the PCM sound data that is to be processed by the processing module(s) 51. The component driver 50 now has direct access to buffer addresses for the PCM sound data destined for the sound card 42. The component driver 50 passes the buffer addresses for the PCM sound data to the one or more processing modules 51. The one or more processing modules 51 effect digital processing of the PCM data, i.e., 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; volume control; and/or data memory storage and then replaces the original buffer PCM data with the processed information. The IRP's are then relayed back to the component driver 50 which, in turn, forwards those IRPs to the port class component driver 34. Alternatively, the IRPs are diverted directly to the port class driver 34 by the one or more processing modules 51.

Principal components of one embodiment of the intercepting component driver 50 constructed in accordance with the present invention are illustrated in Fig. 8. The driver 50 of the illustrated embodiment comprises modules: driver_wdm 50a, driver_wdmDevice 50b, KpnpLowerDevice 50c, tds_driverDevice 50d and tds_DeviceConfig 50f. This particular embodiment of the driver 50 is incorporated into the program listings which are incorporated herein by reference. The module programs perform the following tasks:

driver_wdm

The driver_wdm class contains DriverEntry, LoadRegistryParameters and AddDevice functions. The "DriverEntry" function is called by the operating system when the component driver 50 is loaded into memory. The DriverEntry function inititalizes a Device Object with pointers to various dispatch routines. This functionality is provided by the NuMega framework, noted below, and thus is not explicitly coded in a driver entry routine. The DriverEntry routine also obtains the value of "BreakOnEntry" from the Parameters key section for the driver 50 by invoking the "LoadRegistryParameters" function. In general the AddDevice function is called by the operating system whenever it enumerates a device (i.e., a sound card) related to the driver 50. This function is called when the operating system processes the "UpperFilters" registry entry for the audio device selected. The function first creates a functional device object (FDO). The constructor for the FDO is passed the physical device object (PDO) which represents the driver 38, and binds the newly created FDO to the PDO. This binding causes driver 50 to "see" all IRPs that are directed toward the driver 38. The AddDevice function finishes by checking the status of the FDO and incrementing the unit number if FDO construction is successful.

Functional Device Objects; Physical Device Objects; DriverEntry, LoadRegistryParameters and AddDevice functions; and Device_Contol IRPs, such as 1) KS_PROPERTY 2) KS_WRITE_STREAM are defined and discussed in the following documents: Windows 2000 Driver Development Kit, Microsoft Press, 2000; Programming the Microsoft Windows Driver Model, Walter Oney, Microsoft Press, 1999; Windows NT Device Driver Development, P.G. Viscarola and W. G. Mason, Macmillan Technical Publishing, 1999; Writing Windows WDM Device Drivers, Chris Cant, R&D Books, 1999; Windows 98 Developer's Handbook, Ben Ezzell with Jim Blaney, Sybex, Inc., 1998; The Windows Device Driver Book, Art Baker, Prentice Hall, PTR, 1997; and Windows NT File System Internals, A Developer's Guide, O'Reilly & Associates, 1997.

driver_wdmDevice

This module contains the constructors and destructors for the driver_wdmDevice class as well as numerous dispatch routines for handling IRP flow. The driver_wdmDevice class is derived from the KpnPDevice class provided by software sold by CompuWare® NuMega™ under the product name "DriverStudio, Version 1.0." The constructor for this class calls the base class constructor to create the FDO. It then calls "m_lower.initialize" to bind this FDO to the PDO for the audio device of interest. Since the driver 50 acts as a layered driver for the port class component driver 34 below it, it must be capable of handling any IRP's that might be directed to the port class driver 34. Driver_wdmDevice contains a function entry for all possible IRP's. The majority of these IRP's are simply passed on to the driver 34 below this one since they are of no interest. This is accomplished by calling the "ForceReuseOfCurrentStackLocationInCallDown" method on the IRP followed by calling the "PnpCall" method on the lower driver 34. The IRP_MJ_DEVICE_CONTROL IRP is the IRP of interest. The handler for this type of IRP is "DeviceControl". The evaulation of this IRP is protected by a "spinlock." This method evaluates the IRP for the type needed to effect audio data modification processing and calls the necessary routines to perform the data modification processing. The "privateIO" routine will return a complete status if it handed an IRP associated with data modification control functions, such as setting up a new configuration. In the "complete" case the "spinlock" is released and a return is made by invoking "PnPComplete" on the IRP. In all other cases "privateIO" will return an incomplete status. The DeviceControl function then releases the "spinlock" and passes the IRP on to the lower driver 34.

KPnpLowerDevice

This is a WDM driver class provided by CompuWare® NuMega™ under the product name "DriverStudio, Version 1.0" and is used to represent a Physical Device Object (PDO). When an instance of KPnpLowerDevice is created or initialized by a component driver, it attaches a device object to the Physical Device Object (PDO). The attachment operation returns another system device object address, which points to the device object which was actually attached. This is referred to as the Top Of Stack device. The Top of Stack device may be different from the PDO. Device attachments form a singly linked list, so at most one component driver may attach to a second. If the Top of Stack device is already attached to the PDO, then a second component driver that attempts to attach to the PDO will instead attach to the Top of Stack device. A driver's Functional Device Object (FDO) normally contains a member variable of the KpnpLower device class that is initialized in the AddDevice function to form the linkage between the driver's FDO and the PDO.

tds_driverDevice

tds_DriverDevice is a class containing audio data modification specific functions and members. The primary functions of interest are "privateIO", "dispatchKSProperty", dispatchKSWriteStream", "dispatchTDSSetMode", "dispatchTDSGetMode", "dispatchSetFilter", " dispatchGetFilter", "lock" and "release".
The "privateIO" function vectors processing to routines based on the IoctlCode contained in the IRP. The "dispatchKSProperty" and "dispatchKSWriteStream" functions are responsible for detecting and directing the application program supplied sound buffer to the processing module(s) 51. The "dispatchSetMode" and "dispatchTDSGetMode" are routines which allow the application program to set/get audio modification processing parameters. Those parameters are mode, toggle on , toggle length, scale, and verbose level. The "dispatchSetFilter" and "dispatchGetFilter" routines allow the application program to set/get the digital filter coefficients, which are discussed below. The "lock" and "release" routines are used to set and release a "spinlock". IRPs that are intercepted by tds_driverDevice have their associated PCM data passed by reference to processing module(s) 51.

tds_DeviceConfig

This class represents an enhancement configuration. It contains member variables and accessory functions for configuration name, configuration description, toggle parameters, mode, scale factor, and filter tap settings.

As noted above, the one or more processing modules 51 may effect 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and/or volume control. Activation of the one or more processing modules 51 is effected using the user program 55. This program operates in user mode and preferably includes a graphical user interface. In response to commands input by a user via a keyboard, mouse or other input device, the user interface sends control instructions to the driver 50. Such instructions may specify whether digitized audio data modification should occur and, if so, specify a desired type of digitized audio data modification; and whether digitized audio data should be stored elsewhere in computer memory or on a separate memory device and, if so, if storage should occur before or after audio data modification.

An example user interface is:

1) The User interface program "opens" a channel to the driver using a CreateFile() system call. Specifically it may be in the form

```
Filehandle =CreateFile(device-name,
                GENERIC_READ|GENERIC_WRITE,
                0,
                NULL,
                OPEN_EXISTING,
                FILE_ATTRIBUTE_NORMAL,
                NULL
                );
```

2) Then the user interface examines the returned handle with

```
If(Filehandle != INVALID_HANDLE_VALUE)
{
        //Successful opening of driver
        //now talk to driver using IOCTL commands
}
else
{
        //Failure to open driver
        // Handle the failure
}
```

In the illustrated embodiment, the processing module 51 comprises a digital filter, such as the one described in copending Patent Application, U.S. Serial No. 09/431,371, entitled "Digital Filter," filed on November 1, 1999 or in PCT Patent Application, Serial No. PCT/US00/21912, entitled "Digital Filter," filed on August 10, 2000, the disclosures of which are incorporated herein by reference. The digital filter effects sound enhancement of the PCM sound data. More specifically, the enhancement involves increasing the amplitude of the sound at frequencies where the response of the human ear has been found to drop off.

Media player 10 supplies input digitized sound samples or audio data (i.e., PCM sound data) at a rate which depends upon the nature of the media involved. For a CD, a typical sampling frequency is 44,100 Hz. Many other sampling rates exist. For example, sampling rates of 22,050 Hz and 11,025 Hz are used in many PC applications. Some audio CD's are now produced using other sampling rates. Digital video disks are produced at sampling rates of

64,000 Hz. The telephone industry samples at a rate of 8,000 Hz. The invention described herein is applicable to any sampling rate. The processing module 51 outputs processed digitized audio data at a frequency equal to the rate at which input digitized audio data is supplied by media player 10.

The filter comprises an array of storage locations within a random access memory. A series of N time response values, A_0 through A_{N-1}, are tabulated in a written program for downloading into random access memory storage locations. These response values, hereinafter referred to as time coefficients, are established by a technique described below.

Each digitized sound sample (comprising PCM sound data) is designated X(n) at the time of sampling and is stored temporarily in a designated random access memory location. The sound samples are later shifted successively through a series of N-1 other storage locations for generation of time shifted samples X(n-1) through X(n-[N-1]). Storage locations, may be thought of as being equivalent to stages of a shift register. However, the processing module 51 may not have sufficient memory for this purpose, and therefore the processing module 51 may cause the sound samples to be shifted through a designated portion of the random access memory.

The processing module 51 multiplies each A_n by a corresponding X(n). The $A_nX(n)$ products then are summed. It will be appreciated that the processing module 51 performs N shift operations, N multiplications and N summations, or the equivalent thereof, for each sound sample supplied by the media player 10. IRPs pointing to the calculated sums generated at a rate of 44,100 HZ are supplied to the port class driver 34. In response, the driver 34 provides streams of data having pointers to the processed data to an audio adapter component driver 38, comprising a functional driver, which interfaces directly with the sound card 42. The driver 38 converts the streams of data from the driver 34 into a signal having a format acceptable to the sound card 42. The sound card then retrieves the modified and/or stored digitized audio data from memory and converts the data into an analog signal for driving one or more speakers (not shown).

The processing module 51 convolves N time response coefficients A_0 - A_{N-1} against the entire series of sound samples. Mathematically speaking, the module 51 repeatedly solves the following equation:

$Y(n) = A_0X(n) + A_1X(n-1) + \dots + A_{N-1}X(n-[N-1])$

where A_0 through A_{N-1} are the stored time coefficients;

$X(n)$ is the most recent sample received;

$X(n-1)$ through $X(n-[N-1])$ correspond to N-1 samples received prior to sample $X(n)$;

n is the running index of the time coefficients being computed;

N is equal to the number of terms in the equation to the right side of the equal sign; and

wherein calculated values of Y define the driving signal.

When sampling at 44,100 Hz, all calculations implied by the above equation must be completed within a sampling interval, Δ, of 22.68 microseconds. Actually much less time is available, because the operating system requires time for performing its normal housekeeping functions and running any other applications which simultaneously may be open. Consequently it is necessary to find a series of discrete values for the time response function which are few in number, yet able to produce the high quality sound reproduction desired. In accordance with a preferred embodiment of this invention it has been found that digital music of the desired quality may be generated for an odd value of N (the number of time response coefficients) as low as 7. A noticeable degree of enhancement of digital music has even been accomplished with a value of N as low as 5. Accordingly, it has been found that the value of N is at least 5, preferably 7, more preferably 9, and most preferably 11. It is also contemplated that the value of N may comprise an odd integer greater than 11. It is further contemplated that the value of N may comprise an even integer greater than or equal to 6, such as 8, 10 or 12. However, it is preferred that N comprise an odd integer.

As described above, the convolution process is carried out in real time using appropriate time response coefficients. These time response coefficients are calculated off-line and are mapped to a like number of frequency coefficients established as discussed below. That procedure generates N frequency coefficients H_0-H_{N-1}, each having a frequency and an amplitude, which are mapped into N time response coefficients A_0-A_{N-1} by use of the discrete inverse Fourier transformation:

$$A_n = \frac{1}{N}\sum_{k=0}^{N-1} H_k\, e^{+i2\Pi kn/N}$$

By way of example, N may have a value of 19, which would call for 19 time response coefficients, all derived from a set of 19 frequency coefficients, regularly spaced at intervals F_s in accordance with the time-frequency uncertainty principle, a relationship stated in the following equation:

$$F_s = \frac{1}{\Delta N}$$

Thus for $\Delta = 1 / 44{,}100$ and $N = 19$, $F_s = 2321$ Hz. The results are set forth in TABLE I below.

Table I

Freq. ~ Hz	Amp of Freq Coeff	Amp of Time Coeff	Phase Angle of Time Coeff
0	2.5	3.0295	0
2321	2.5	0.4046	Π
4642	2.15	0.01	Π
6963	2.4	0.1905	0
9284	2.6	0.0237	Π
11605	2.9	0.0694	Π
13926	3.78	0.2137	0
16247	4.6	0.2074	Π
18568	2.6	0.1175	0
20889	4	0.0714	Π
23210	4	0.0714	Π
25531	2.6	0.1175	0
27852	4.6	0.2074	Π
30173	3.78	0.2137	0
32494	2.9	0.0694	Π

34815	2.6	0.0237	Π
37136	2.4	0.1905	0
39457	2.15	0.01	Π
41778	2.5	0.4046	Π

In general the A_n and the H_k are all complex numbers, having a magnitude and a phase. However, in the practice of an embodiment of this invention the H_k have a phase angle of 0, and the A_n have a phase angle of either 0 or Π. Values of Π (i.e., 3.14159) are accommodated in the filter design by assigning a negative value to the amplitude of the time coefficient. The 19 frequency domain points define a frequency response curve 58 extending from 0 Hz to 41,778 Hz as illustrated in Fig. 5. They include a base point 60, principal points 61 - 69 and mirror points 71 - 79. The specified numeric amplitudes of the various frequency domain points 60-69 and 71-79 do not have units. The specified amplitudes of these points are all relative to one another. Frequency domain points 60 – 69 and 71 - 79 are set at regular intervals of 2321 Hz with mirror points 71 - 79 having amplitudes equal to principal points 61 - 69 respectively. It will be observed that the principal points and the mirror points are mirrored about a mid frequency of 22,050 Hz, half the 44,100 Hz sampling frequency. This produces a periodic response at the sampling frequency. That periodicity effectively extends the frequency response curve 58 to 44,100 Hz and sets a virtual point (not illustrated) at 44,100 Hz having a magnitude equal to that of base point 60. This virtual frequency domain point is mapped to a virtual time response point (also not illustrated) equal to, and synchronous with, A_0. Since 19 is an odd number there is no frequency domain point at the mirroring mid frequency.

Since the human ear does not typically respond to frequencies in excess of about 20,000 Hz, the entire information content of the original audio signal can be processed by low pass filtering at 22,050 Hz, followed by sampling and recording at 44,100 Hz. CDs and other digital recordings are usually prepared in this fashion. That gives two points per cycle of the highest frequency present.

Frequency response points were selected in the range from 20 Hz to 20,000 Hz having amplitudes bearing a roughly inverse relationship to the sensitivity of the human ear over that same range. This flattens out the perceived response of the filter to the regenerated sound. In particular the frequency response was relatively decreased at points 68 and 62 to compensate for regions of increased ear sensitivity near 4642 Hz and 18,568 Hz respectively, and it was relatively increased at point 63 to compensate for reduced ear sensitivity in the region around 16,247 Hz. Thus, as is apparent from Fig. 5, frequency response coefficients 62-69 having frequencies between a reference frequency of 4642 Hz and a high end frequency of 20,000 Hz increase in amplitude as per increasing frequencies from the reference frequency up to a significant amplitude peak at a peak high frequency of 16,247 Hz and decrease in amplitude as per increasing frequencies down to a significant amplitude trough at a trough high frequency of 18,568 Hz. The amplification of the frequency response coefficient at the peak high frequency is about 2.1 times the amplification of the frequency response coefficient at the reference frequency. The frequency response coefficient 69 which is positioned between the reference frequency and a low end frequency of 20 Hz has an amplitude which is slightly greater than that of the frequency response coefficient 68 located at the reference frequency.

The required filter response is undefined outside the range of human hearing. That includes the region below about 20 Hz, a free band region between 20,000 Hz and 24,100 Hz, and an end region between 24,100 Hz and 44,100 Hz. However, the periodic nature of the filtering process requires some definition for all frequencies between zero and the sampling frequency. Therefore, frequency response points or taps were selected at amplitudes which would tend to lend smoothness to the response function in the undefined regions. Hence, point 60 having a frequency of 0 Hz was selected so as to have an amplitude of 2.5, which is equal to the amplitude for point 69, which has a frequency of 2321 Hz. The amplitudes of points 61 and 71, which fall within the free band region, were selected so as to achieve a generally constant oscillation frequency across a center band extending from about 11,605 Hz to about 32,494 Hz, i.e., the portions of the curve defined between points 65,62; 62,72; and 72,75 have approximately the same width. Points 72-79 are mirror images of points 62-69. By selecting the amplitudes of points 60, 61 and 71-79 in the manner discussed above, the number of time coefficients having significant or large magnitudes were reduced. In order to reduce processing time during filtering, the number of time coefficients incorporated into the filter must be minimized. By minimizing the number of time coefficients having significant magnitudes, remaining time coefficients having insignificant magnitudes may be discarded resulting in fewer time coefficients required to achieve a desired frequency response for the filter.

The above described digital filter design procedure was found to produce time response coefficients which, when convolved with typical stored music, rendered output sound of remarkably high quality. It was found that background sound which is ordinarily suppressed during storage, was substantially restored upon playback. Moreover, 19 coefficients did not overwhelm the computer. Having achieved those results, filters having lower values of N were then designed.

The following is a general summary of how a frequency response curve according to the present invention can be affected by manipulating various frequency response points or taps. Also included are some of the characteristics that can occur when manipulating specific taps located in each region of the frequency response curve noted. In general, manipulation of one or more of the taps at frequencies lower than the reference frequency can affect the low frequency response or base region of the frequency response curve (e.g., about 0 Hz to about 2004 Hz). By raising the amplitude of one or more of the taps in the base region, the base of the resulting sound can be made to sound more boomy and thick (i.e., the base sound is over emphasized and it loses some of its detail and clarity). Raising the amplitude of one or more of the base region taps also seems to affect the high frequency response or treble region (above the reference frequency) so that the treble portion sounds thicker and not quite as brilliant (i.e., it loses detail and clarity). That is, affecting the base region in this manner can suppress the treble region in a manner similar to the high frequency masking effect caused by using noise suppression technology. When the amplitude of one or more of the taps in the base region are lowered, the base of the resulting sound can be made to sound much thinner, which can result in the treble region sounding brighter or brittle (i.e., more apparent or over emphasized), depending upon how the taps are adjusted.

In general, manipulation of one or more of the taps within the range of the reference frequency (e.g., about 501 Hz to about 8018 Hz) can affect the reference frequency response of the frequency response curve. Raising the amplitude of one or more of the taps in this region can produce sound appearing to have more air and greater separation between the sound sources, e.g., instruments (i.e., different sound sources are more easily distinguished). This greater separation can make the overall sound appear more like it was live and not recorded. Lowering the amplitude of one or more of the taps in this region can make the resulting signal sound dull and lifeless.

Manipulation of one or more of the taps at frequencies higher than the reference frequency, but within the frequency band "of interest", can affect the high frequency response or treble region of the frequency response curve (e.g., about 2004 Hz to about 20045 Hz). By raising the amplitude of the taps in the treble region, more brilliance can be added to the resulting sound. In this way, a greater amount of detail can be offered, especially in audio application like, for example, a movie soundtrack. By lowering the amplitude of the taps in the treble region, the resulting sound can be made to sound flat and muted or muddy.

Manipulation of one or more of the taps out beyond the frequency band "of interest" (i.e., the center region where the curve is folded upon itself) can affect the dwell time of the signal, especially when the frequencies of the taps are at the edge or above the band of normal human hearing (e.g., about 20 Hz to about 20,000 Hz). For example, the center region of the frequency response curve of Fig. 6 contains taps 101 and 111. By raising the amplitude of the taps in this region, the dwell time can be reduced and the resulting enhancement can sound more like a mere amplitude shift, rather than overall harmonic enhancement. Increasing the amplitude of these taps can also increase the rate of clipping or overdriving of a signal. In addition, by lowering the amplitude of the taps in this region, the dwell time can be increased and the overall enhancement and stability of the audio signal improved, with a reduction in the risk of the signal being clipped. The dwell time can be dramatically affected by changes in the amplitude of the center region taps, regardless of the number of the center region taps. The rate of transition between the frequency band of interest and the center region can be controlled by adjusting the amplitude of one or more of the center region taps. Surprisingly, it has been found that the cutoff characteristic between the frequency band of interest and the center region (i.e., the sharpness with which the band edge is formed) can be used to control the dwell time and, consequently, the enhancement imparted to the digital audio signal. As the dwell time increases, certain frequencies exhibit an increase in loudness (as sounds are stretched in time they appear to be louder). This increase in "dwell time" is responsible for unmasking previously masked tones and leads to a preferred enhancement. Dwell time is defined here as the time interval over which the impulse response of the filter has significant amplitude. For a filter with a fixed number of taps (i.e., a finite impulse response filter), the actual time duration of the impulse response cannot be modified.

A digital filter in accordance with a second embodiment of the present invention was created as follows. A set of first frequency response coefficients separated at uniformly spaced frequency intervals were selected. The first frequency response coefficients had zero phase angles. In the illustrated embodiment, the frequency response coefficients set out in Table 1 above were used as the first frequency response coefficients. By the use of discrete inverse Fourier transformation, the plurality of first frequency response coefficients were mapped into corresponding first time response coefficients, which are also set out in Table 1.

As noted above, in order to reduce processing time during filtering, the number of time coefficients should be minimized. So as to achieve that end, a pair of the first time response coefficients having equal magnitudes and being positioned adjacent to one another in Table 1 were removed. The discarded pair of time coefficients each had a magnitude of .0714. The remaining time coefficients defined second time coefficients.

If an even number of first time response coefficients are provided, the one first time response coefficient not having a zero frequency and not having a mate of equal magnitude is also discarded.

The effect of removing the pair of time coefficients on the frequency response of the digital filter was then assessed. This involved generating a first frequency response curve from the first frequency response coefficients. The second time response coefficients were mapped into corresponding second frequency response coefficients by use of the discrete Fourier transformation:

$$H_k = \sum_{n=0}^{N-1} A_n \, e^{-i\frac{2\Pi kn}{N}}$$

A second frequency response curve was then generated using the second frequency response coefficients. The first and second frequency response curves were compared to determine if the second frequency response curve was substantially different from the first frequency response curve. If not, then another pair of time response coefficients, i.e., a pair of the second time response coefficients, having equal magnitudes and being positioned adjacent to one another were removed. This process of removing and assessing continued until a pair of discarded time response coefficients caused a significant change in the perceived enhancement caused by the frequency response of the digital filter. A significant change in the perceived enhancement caused by the frequency response of the digital filter corresponds to a substantial difference between an initial and a subsequent frequency response curve. When a significant change occurred, this last pair of discarded time response coefficients were added back to the time response coefficients. These remaining time response coefficients defined final time response coefficients.

In order to reduce processing time, it is preferred that the time response coefficients comprise integer numbers. In order to convert the final time response coefficients into integers, they were multiplied by an integer conversion number sufficiently large to permit any remaining fractional portion to be discarded without losing substantial final time response coefficient accuracy. A substantial loss in final time response coefficient accuracy occurs when the resulting frequency response of the digital filter produces an enhancement perceptibly different from the desired enhancement. It is also preferred that the integer conversion number be selected as a power of two so that shifting may be used in place of division when subsequent renormalization occurs for calculated values of Y. In the illustrated embodiment, the final time coefficients (set out in brackets in Table 2 below) were multiplied by 2^{13} (8192). Any remaining fractional portions of the converted time coefficients were discarded. The integer final time coefficients are set out in Table 2 below. They may be tabulated in a written program for downloading into random access memory storage locations. This filter has 11 time response coefficients.

Prior to the microprocessor supplying the calculated values of Y to the audio adapter component driver 38, the microprocessor must renormalize those values. This involves dividing each value of Y by the integer conversion number. Alternatively, if the integer conversion number is selected as a power of two, the microprocessor can effect renormalization by right shifting an appropriate number of bit positions, 13 in the illustrated embodiment.

A frequency response curve 80 plotted from the final frequency response coefficients, set out in Table 2, is shown in Fig. 6. Curve 80 has a base point 100. It also has principal points 101 - 105, which are mirrored by points 111- 115. As is apparent from Fig. 6, frequency response coefficients 102-105 having frequencies between a reference frequency of 4009 Hz and a high end frequency of 20,000 Hz increase in amplitude as per increasing frequencies from the reference frequency toward the high end.

TABLE 2

Freq. ~ Hz	Amp of Freq Coeff	Amp of Time Coeff	Phase of Time Coeff
00000	2.50	23,980 (2.9273)	0
04009	2.00	3,593 (.4386)	Π
08018	2.40	125 (0.0153)	0
12027	2.85	1226 (0.1497)	0
16036	4.00	598 (0.0731)	Π
20045	3.50	1089 (0.1330)	0
24054	3.50	1089 (0.1330)	0
28063	4.00	598 (0.0731)	Π
32072	2.85	1226 (0.1497)	0
36081	2.40	125 (0.0153)	0
40090	2.00	3,593 (0.4386)	Π

A filter in accordance with a third embodiment of the present invention was created using the same design procedures undertaken to construct the filter of the second embodiment. The third filter's final frequency and time response coefficients are set out in Table 3 below. The final time response coefficients both before and after being multiplied by an integer conversion number equal to 8192 are set out in Table 3. A frequency response curve 90 plotted using the final frequency response coefficients is shown in Fig. 7. The curve has a base point 120, principal points 121-125 and mirror points 131-135. As is apparent from Fig. 7, frequency response coefficients 122-125 having frequencies between a reference frequency of 4009 Hz and a high end frequency of 20,000 Hz increase in amplitude as per increasing frequencies from the reference frequency toward the high end.

TABLE 3

Freq. ~ Hz	Amp of Freq Coeff	Amp of Time Coeff	Phase of Time Coeff
00000	2.50	24,948 (3.0455)	0
04009	1.00	5958 (0.7274)	Π
08018	2.50	1634 (0.1995)	Π
12027	3.50	3094 (0.3777)	0
16036	5.00	237 (0.0290)	Π
20045	3.50	2503 (0.3056)	0
24054	3.50	2503 (0.3056)	0
28063	5.00	237 (0.0290)	Π
32072	3.50	3094 (0.3777)	0
36081	2.50	1634 (0.1995)	Π
40090	1.00	5958 (0.7274)	Π

The above description of the invention teaches a straight forward calculation of the sound card driving signal, Y. A more preferred procedure takes advantage of an observed symmetry in the time coefficients. That is, $A_m = A_n$ where $0 < n \leq (N + 1) / 2$ and $m = N - n$. In the case where N = 11 a simplified calculation of the sound card driving signal takes the form:

$$Y = A_0X_0 + A_1(X_1 + X_{10}) + A_2(X_2 + X_9) + A_3(X_3 + X_8) + A_4(X_4 + X_7) + A_5(X_5 + X_6)$$

This reduces the number of time consuming multiplications. The procedure may be extended to other values of N.

Filters constructed in accordance with the present invention may have a reference frequency which falls within the range of from about 501 Hz to about 8018 Hz; a peak high frequency which falls within the range of from about 1002 Hz to about 20045 Hz; a trough high frequency which can fall at any frequency after the peak high frequency; and a peak low frequency which falls within the range of from about 0 Hz to about 2004 Hz. It is further contemplated that the amplification of the frequency response coefficient at the peak high frequency may be about 1.3 times to about 6.0 times the amplification of the frequency response coefficient at the reference frequency. It is also contemplated that the amplification of the frequency response coefficient at the peak low frequency may be about 1.1 times to about 3.0 times the amplification of the frequency response coefficient at the reference frequency.

It is additionally contemplated that two or more of the filters described herein may be stored in a single processing module 51 or separately in two or more processing modules 51. Hence, a user may select a desired one of a plurality of filters for use in enhancing digitized audio data via the user program 55.

It is further contemplated that designer software may be used in conjunction with the driver 50 and the processing module(s) 51 so as to permit a designer to develop a new filter for incorporation into the driver 50, an existing a processing module 51 or a new processing module 51. The software comprises a graphical interface having a set of controls representing frequency sample points in the audio frequency range (i.e., 20 Hz - 20 kHz). The software allows the designer to set the frequency sample points as desired to obtain a given frequency compensation or response curve. The software then calculates the temporal sample points that correspond to the time coefficients required for the real-time linear filtering of digitized audio data and places those temporal coefficients into either the driver 50 or a processing module 51 by communicating with the driver 50 or the processing module 51 through IOCTL calls. The software further functions to give a designer-defined name to the new filter and communicate same to the user program 55. The new filter then becomes available to a user via the user program 55.

What is claimed is.

Claims

1. A device driver forming part or an extension of an operating system, the operating system being adapted to receive digitized audio data from an application program, the device driver comprising: an intercepting component driver which is capable of initiating at least one of sound modification processing of the digitized audio data and storage of the digitized audio data.

2. A device driver as set forth in claim 1, wherein the component driver effects sound modification processing of the digitized audio data.

3 A device driver as set forth in claim 2, wherein the sound modification processing comprises at least one of 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and volume control.

4. A device driver as set forth in claim 2, wherein the sound modification processing comprises sound enhancement processing using a digital filter, the digital filter comprising a series of digitized time coefficients stored in a memory, the time coefficients being mapped to a like number of frequency coefficients, the frequency coefficients being spaced at frequency intervals, having either zero phase angles or linearly spaced phase angles and having amplitudes which are mirrored about a mid frequency to produce periodicity of a time response for the digital filter.

5. A device driver as set forth in claim 4, wherein the time coefficients are odd in number.

6. A device driver as set forth in claim 5, wherein the number of the time coefficients is equal to or greater than 5.

7. A device driver as set forth in claim 4, wherein the time coefficients are defined by inverse discrete Fourier transforms of the frequency coefficients.

8. A device driver as set forth in claim 4, wherein a portion of the frequency coefficients having frequencies within a free band are selected so as to achieve a generally constant oscillation frequency across a center band which is broader than said free band.

9. A device driver as set forth in claim 4, wherein the frequency coefficients are spaced at equal frequency intervals.

10. A device driver as set forth in claim 4, wherein the time coefficients are integer numbers.

11. A device driver as set out in claim 2, wherein the component driver further initiates storage of the digitized audio data on one of a compact disk, a digital video disk, a floppy disk and computer memory.

12. A device driver as set forth in claim 1, wherein the component driver initiates storage of the digitized audio data on one of a compact disk, a digital video disk, a floppy disk and computer memory.

13. A device driver as set forth in claim 1, wherein the component driver initiates the at least one of sound modification processing of the digitized audio data and storage of the digitized audio data in kernel mode.

14. A device driver forming part or an extension of an operating system, the operating system being adapted to receive digitized audio data from an application program, the device driver comprising:
at least two layered component drivers including a functional component driver which manages the operation of a sound card device and an intercepting component driver positioned above the functional driver and being capable of initiating at least one of sound modification processing of the digitized audio data and storage of the digitized audio data.

15. A device driver as set forth in claim 14, wherein the digitized audio data is stored in a first

portion of computer memory and the intercepting component driver initiates the at least one of sound modification processing and storage by intercepting a signal pointing to the digitized audio data stored in the first portion of computer memory and output by the application program, retrieving the digitized data from the first portion of memory and effecting the at least one of sound modification processing of the digitized data and storage of the digitized data.

16. A device driver as set forth in claim 15, wherein the intercepting component driver effects sound modification processing of the digitized data.

17. A device driver as set forth in claim 16, wherein the intercepting component driver effects the sound modification of the digitized audio data via at least one of 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and volume control.

18. A device driver as set forth in claim 17, wherein the intercepting component driver effects sound enhancement processing of the digitized audio data using a digital filter, the digital filter comprising a series of digitized time coefficients stored in a memory, the time coefficients being mapped to a like number of frequency coefficients, the frequency coefficients being spaced at frequency intervals, having either zero phase angles or linearly spaced phase angles and having amplitudes which are mirrored about a mid frequency to produce periodicity of a time response for the digital filter.

19. A device driver as set forth in claim 16, wherein the intercepting component driver further causes the digitized audio data to be stored on one of a compact disk, a digital video disk, a floppy disk and a second portion of computer memory.

20. A device driver as set forth in claim 15, wherein the intercepting component driver causes the digitized audio data to be stored on one of a compact disk, a digital video disk, a floppy disk and a second portion of computer memory.

21. A device driver as set forth in claim 14, wherein the digitized audio data is stored in a first portion of computer memory and the intercepting component driver initiates the at least one of sound modification processing and storage by intercepting a signal pointing to the digitized audio data stored in computer memory and output by the application program, transferring the signal to a separate software module such that the module may retrieve the digitized data from the first portion of memory and effect the at least one of sound modification processing of the digitized data and storage of the digitized data.

22. A device driver as set forth in claim 21, wherein the software module effects sound modification processing of the digitized data.

23. A device driver as set forth in claim 22, wherein the software module effects the sound modification of the digitized audio data via at least one of 3-dimensional processing; reverberation processing; equalization processing; sound enhancement processing; and volume control.

24. A device driver as set forth in claim 23, wherein the software module effects sound enhancement processing of the digitized audio data using a digital filter, the digital filter comprising a series of digitized time coefficients stored in a memory, the time coefficients being mapped to a like number of frequency coefficients, the frequency coefficients being spaced at frequency intervals, having either zero phase angles or linearly spaced phase angles and having amplitudes which are mirrored about a mid frequency to produce periodicity of a time response for the digital filter..

25. A device driver as set forth in claim 22, wherein the software module further causes the digitized audio data to be stored on one of a compact disk, a digital video disk, a floppy disk and a second portion of computer memory.

26. A device driver as set forth in claim 21, wherein the software module causes the digitized audio data to be stored on one of a compact disk, a digital video disk, a floppy disk and a second

portion of computer memory.

27. A device driver as set forth in claim 14, wherein the component driver initiates the at least one of sound modification processing of the digitized audio data and storage of the digitized audio data in kernel mode.

28. A computer-implemented method of creating processed sound from digitized audio data, said method comprising the steps of:

storing said digitized audio data in memory;

creating a kernel mode streaming signal pointing to said stored audio data;

using said kernel mode streaming signal for kernel mode enhancement of said data, said kernel mode enhancement comprising real time inverse convolution of said audio data and a series of time coefficients derived from frequency coefficients inversely related to the response of the human ear across a range of frequencies; and

using said audio data, enhanced as aforesaid, for kernel mode generation of processed sound.

29. A method according to claim 28, wherein said enhancement of said audio data is performed by an intercepting component driver in a device driver or a separate software module.

30. A method according to claim 28, wherein the steps are preformed in a digital computer having at least a 32-bit Windows™ operating system.

31. A method of processing sound comprising the steps of:

1. generating a set of frequency coefficients inversely related to the responsiveness of the human ear,
2. converting said frequency coefficients to time coefficients,
3. sampling said sound to generate an endless series of raw audio data,
4. storing said raw audio data,
5. generating a series of addresses indicating the locations of the stored data,

6. reading said raw audio data,

7. generating a series of processed audio data by convolving said raw audio samples against said time coefficients, and

8. using said processed audio data to drive a sound generator.

32. A method according to claim 31 wherein said steps 4 through 7 are performed by a digital computer operating in kernel mode.

FIG. 1 PRIOR ART

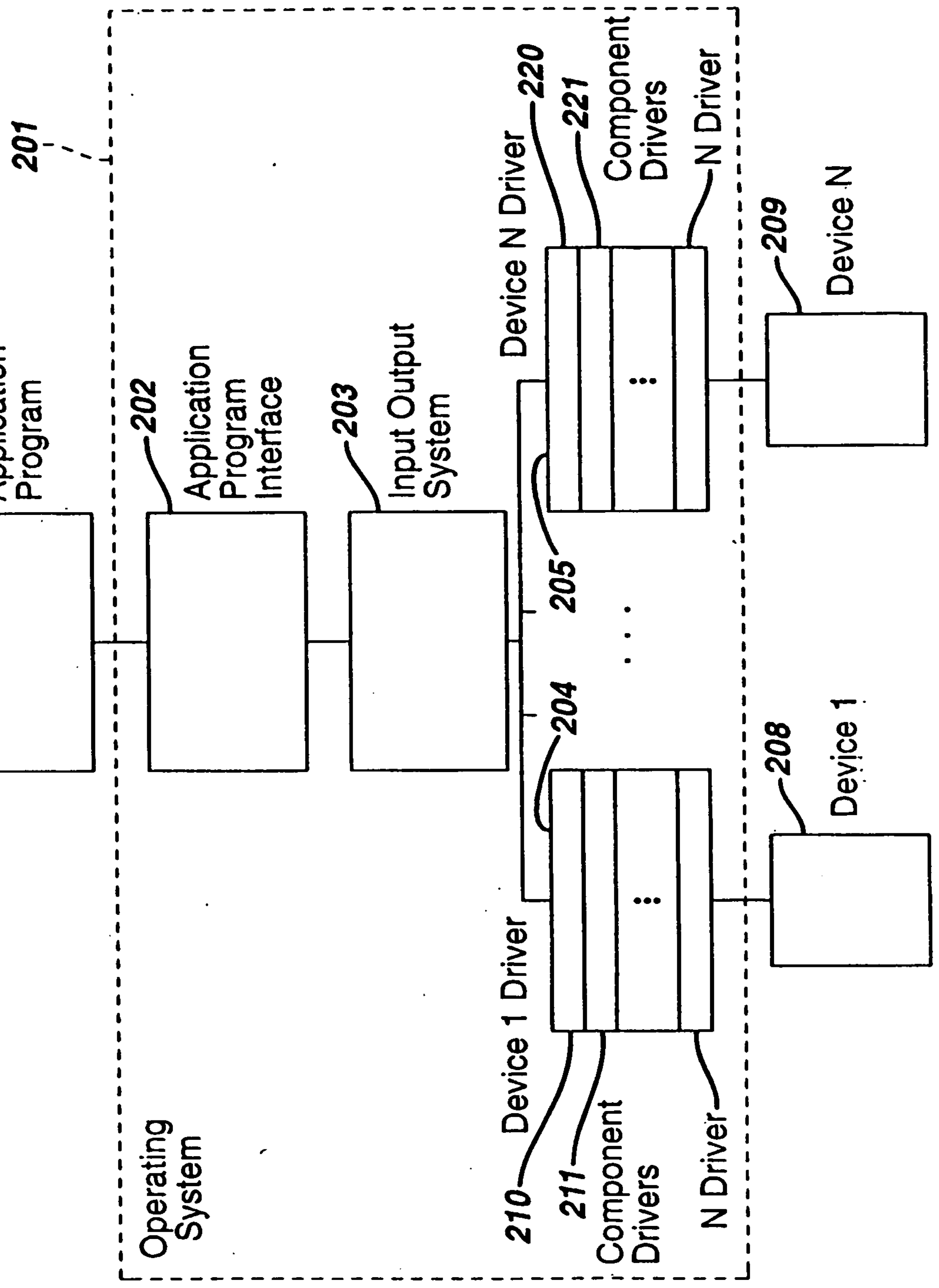


FIG. 1A PRIOR ART

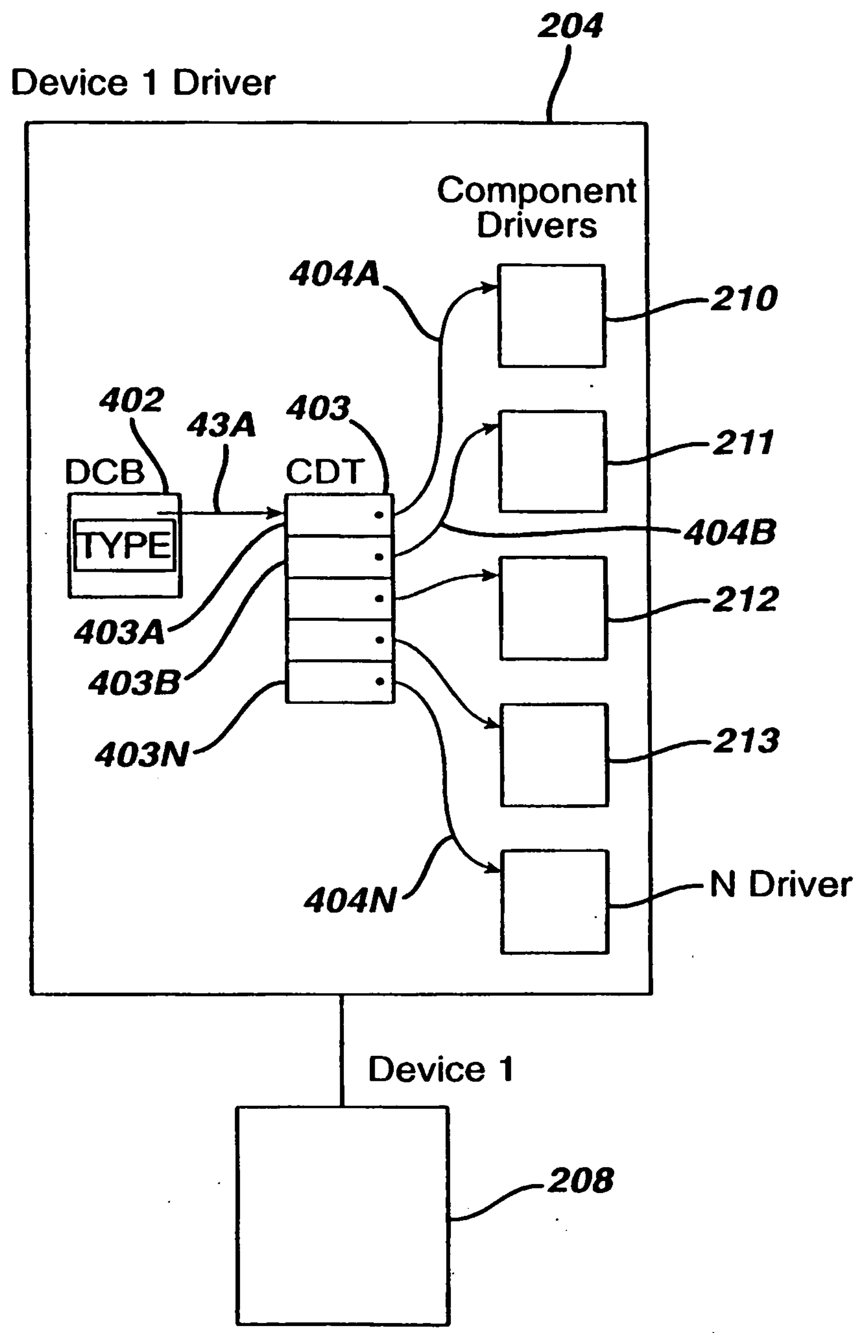


FIG. 2

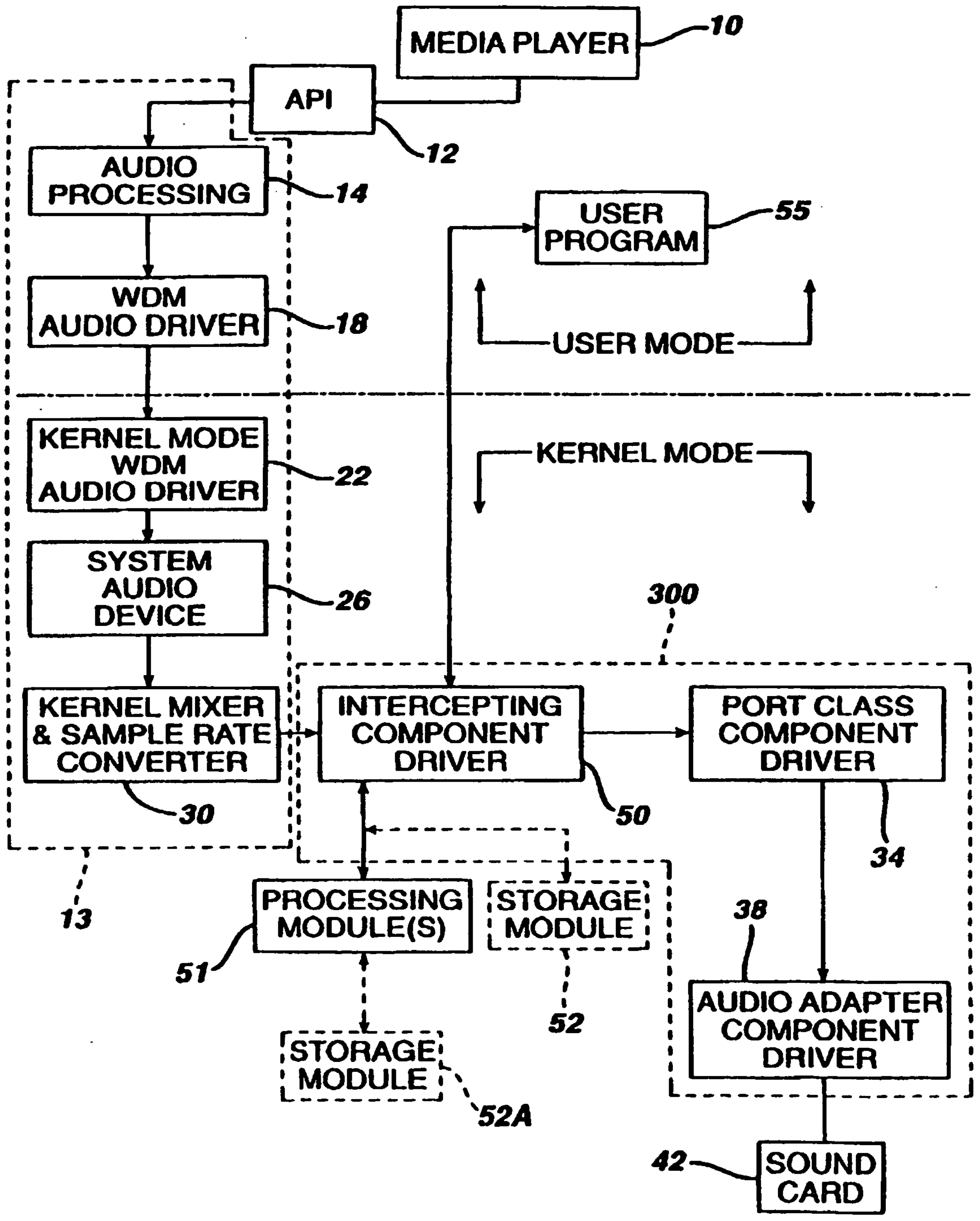

MEDIA PLAYER
10
API
12
AUDIO PROCESSING
14
USER PROGRAM
55
WDM AUDIO DRIVER
18
USER MODE
KERNEL MODE WDM AUDIO DRIVER
22
KERNEL MODE
SYSTEM AUDIO DEVICE
26
300
KERNEL MIXER & SAMPLE RATE CONVERTER
INTERCEPTING COMPONENT DRIVER
PORT CLASS COMPONENT DRIVER
30
50
34
13
PROCESSING MODULE(S)
STORAGE MODULE
38
51
52
AUDIO ADAPTER COMPONENT DRIVER
STORAGE MODULE
52A
42
SOUND CARD

FIG. 3

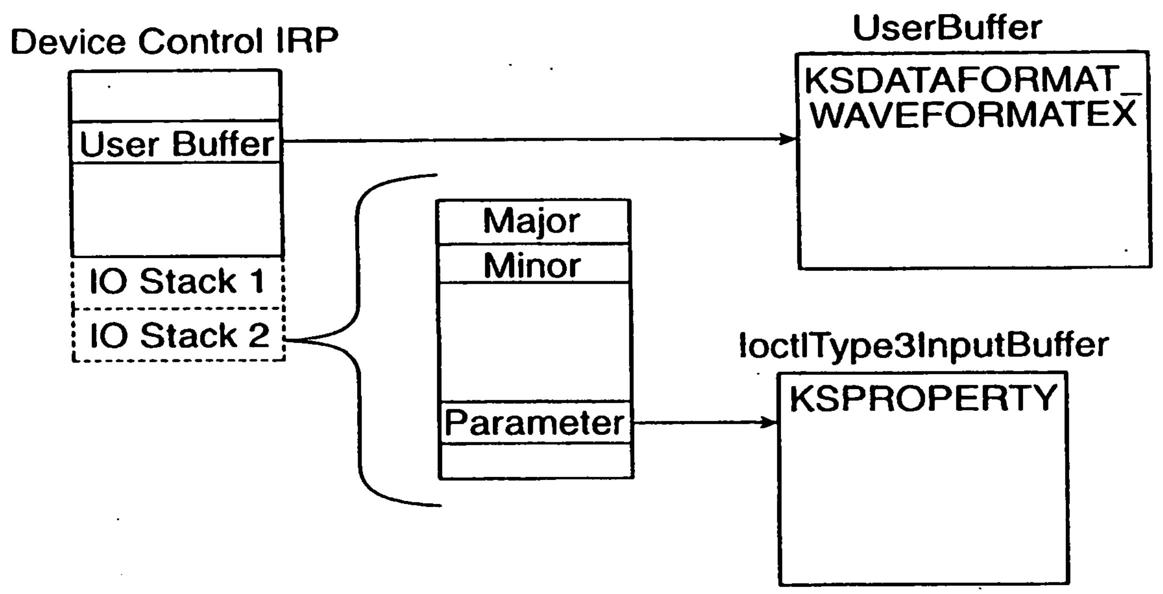

Device Control IRP
User Buffer
IO Stack 1
IO Stack 2
Major
Minor
Parameter
UserBuffer
KSDATAFORMAT_
WAVEFORMATEX
IoctlType3InputBuffer
KSPROPERTY

FIG. 4

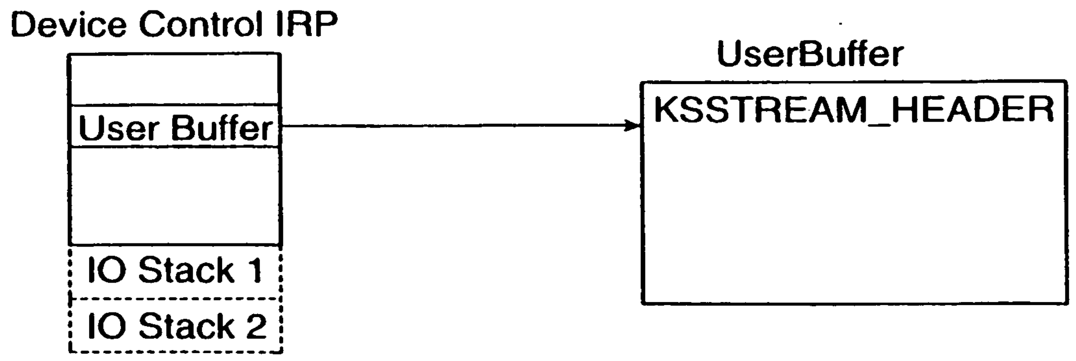

Device Control IRP
User Buffer
IO Stack 1
IO Stack 2
UserBuffer
KSSTREAM_HEADER

FIG. 5

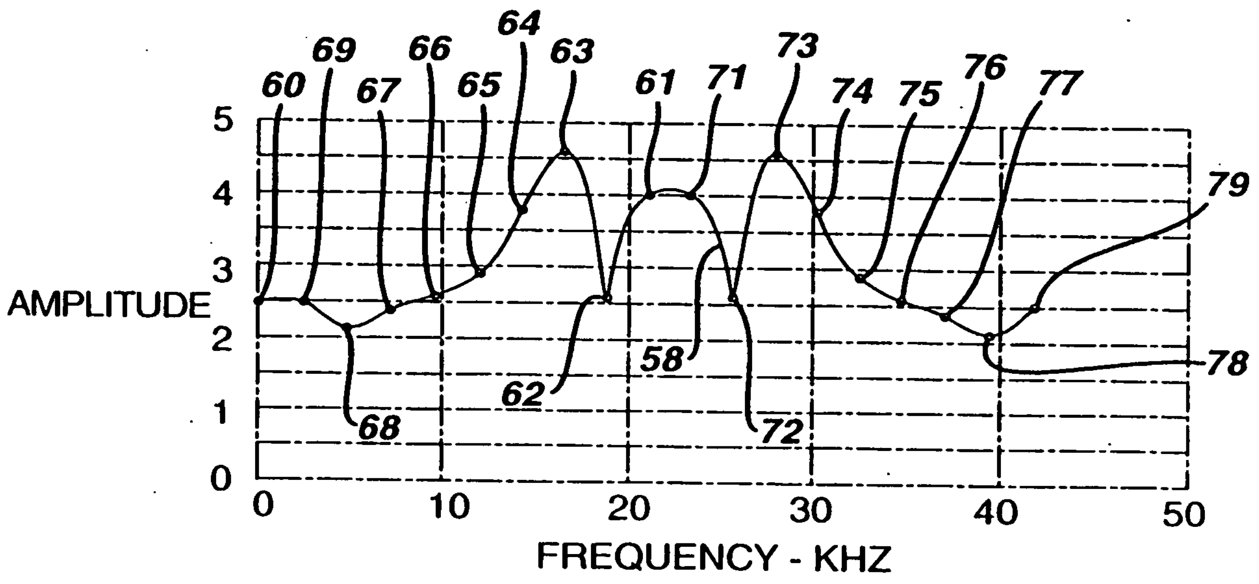

60
69
67
66
65
64
63
61
71
73
74
75
76
77
79
78
68
62
58
72
5
4
3
2
1
0
AMPLITUDE
0
10
20
30
40
50
FREQUENCY - KHZ

FIG. 6

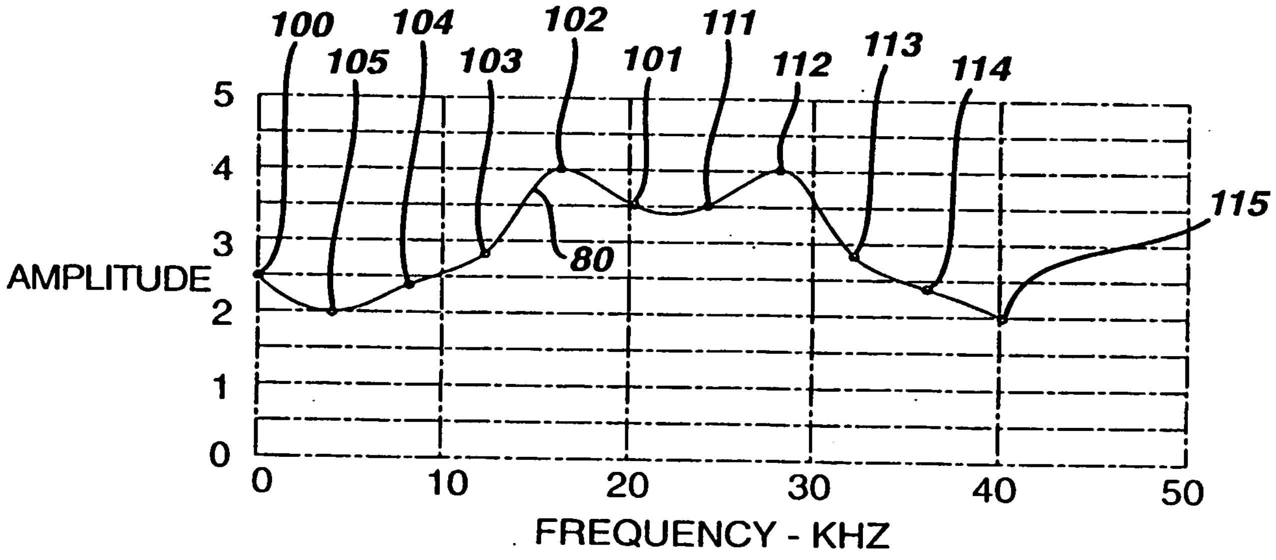

100
105
104
103
102
101
111
112
113
114
115
80
5
4
3
2
1
0
AMPLITUDE
0
10
20
30
40
50
FREQUENCY - KHZ

FIG. 7

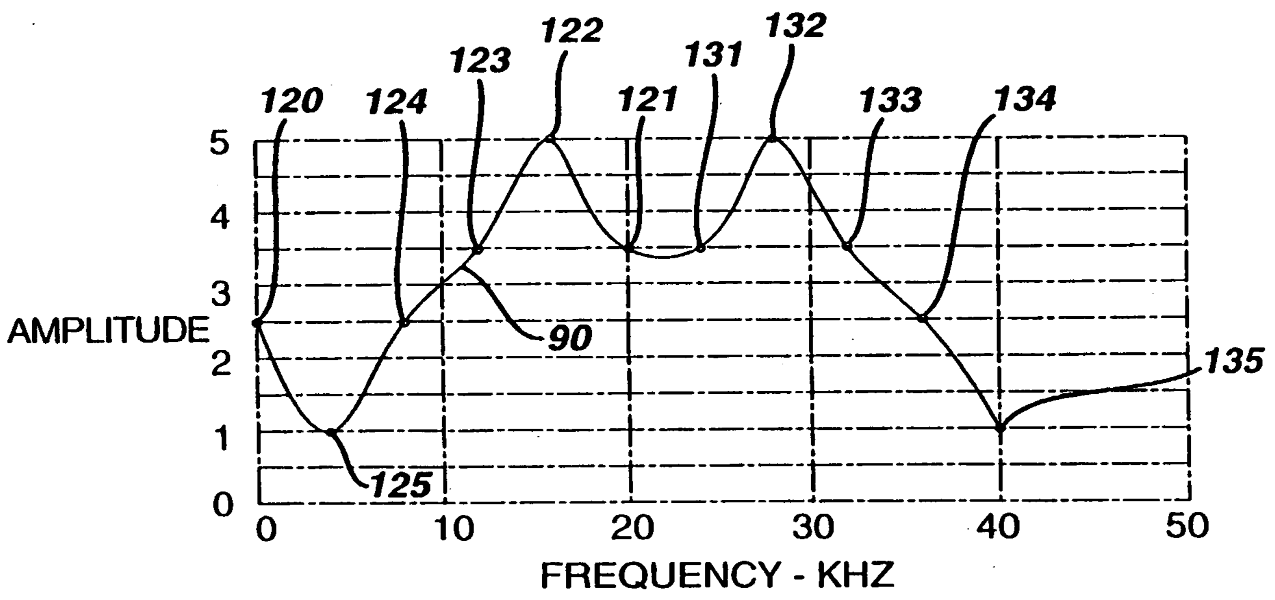
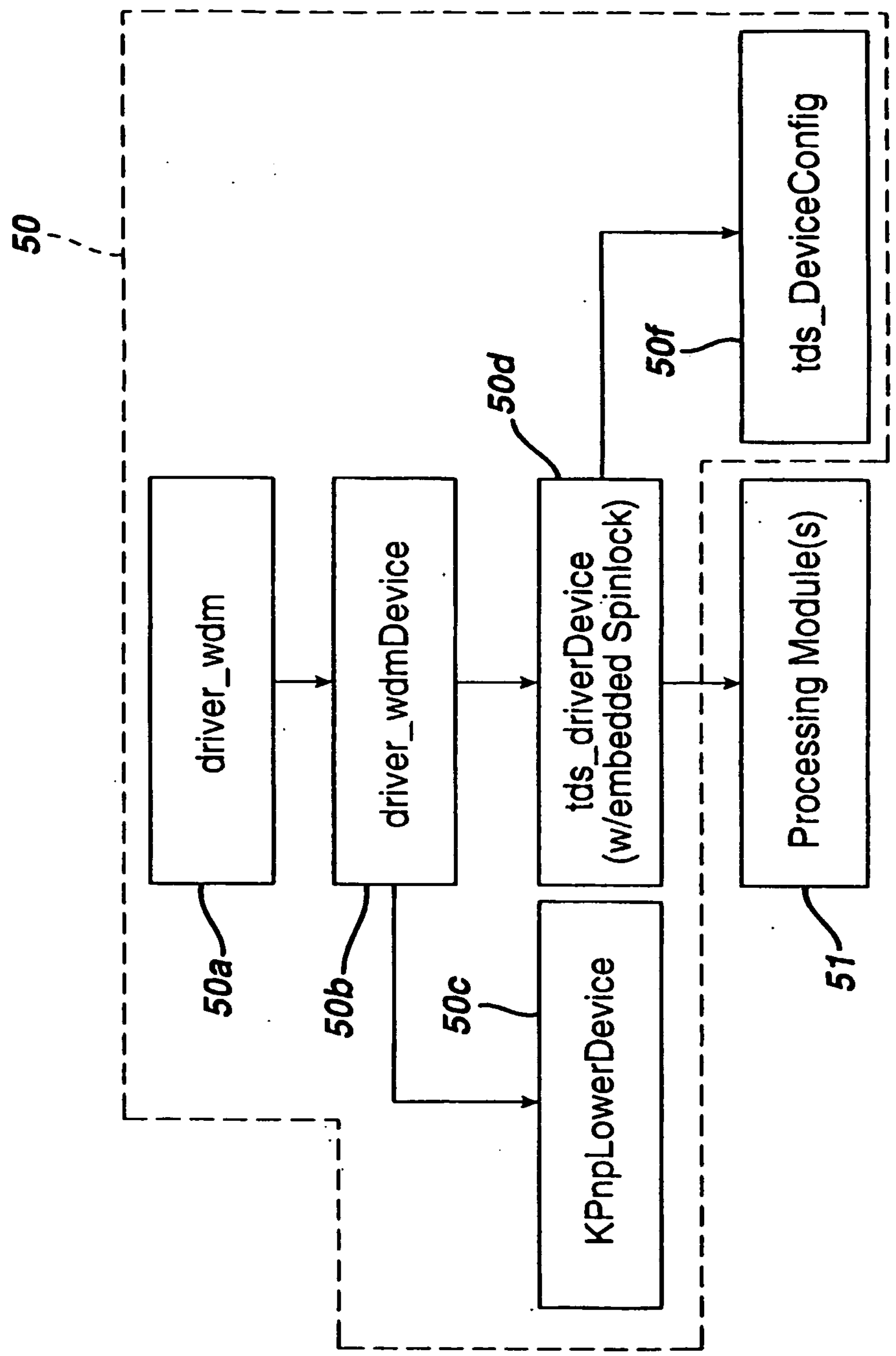

120
124
123
122
121
131
132
133
134
135
125
90
AMPLITUDE
FREQUENCY - KHZ
0
1
2
3
4
5
0
10
20
30
40
50

FIG. 8


50
driver_wdm
50a
driver_wdmDevice
50b
50c
KPnpLowerDevice
50d
tds_driverDevice (w/embedded Spinlock)
50f
tds_DeviceConfig
Processing Module(s)
51